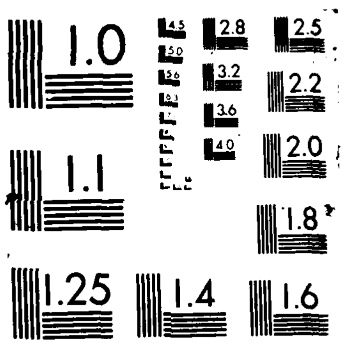
AD-A186 370 C3EVAL MODEL DEVELOPMENT--1986 VOLUME 2 PROGRAMMERS' MANUAL(U) INSTITUTE FOR DEFENSE ANALYSES ALEXANDRIA VA R F ROBINSON ET AL. APR 87 IDA-P-1978-VOL-2 1/4

UNCLASSIFIED IDA/HQ-86-31573 MDA903-84-C-0031 F/G 12/5 NL


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

DTIC FILE COPY

Copy 54 of 75 copies

(2)

AD-A186 370

IDA PAPER P-1978

C^{3}EVAL MODEL DEVELOPMENT -- 1986

Volume II: Programmers' Manual

Robert F. Robinson, *Project Leader*
M. L. Roberson (Applications Research Corporation)
D. W. Roberson (Applications Research Corporation)

April 1987

Prepared for
Joint Chiefs Of Staff

IDA
INSTITUTE FOR DEFENSE ANALYSES
1801 N. Beauregard Street, Alexandria, Virginia 22311

IDA Log No. HQ 86-31573

87 9 18 [illegible]

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

AD-A186 370

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY DD Form 254 dated 1 October 1983	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE NA	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) IDA Paper P-1978	5. MONITORING ORGANIZATION REPORT NUMBER (S)
6a. NAME OF PERFORMING ORGANIZATION Institute for Defense Analyses 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION OUSDRE (DoD-IDA Management Office)
6c. ADDRESS (CITY, STATE, AND ZIP CODE) 1801 North Beauregard Street Alexandria, Virginia 22311	7b. ADDRESS (CITY, STATE, AND ZIP CODE) 1801 North Beauregard Street Alexandria, Virginia 22311
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Joint Chiefs of Staff (J-6) 8b. OFFICE SYMBOL	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER MDA 903 84C 0031
8c. ADDRESS (City, State, and Zip Code) The Pentagon Washington, DC 20301-5000	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT: — PROJECT NO.: — TASK NO.: T-I6-309 — ACCESSION NO. WORK UNIT: —

11. TITLE (Include Security Classification)
C3EVAL MODEL DEVELOPMENT--1986, Volume II: Programmers' Manual

12. PERSONAL AUTHOR(S).
Robert. F. Robinson, M. L. Roberson (ARC) and D. W. Roberson (ARC)

13. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM TO	May 1987	328

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Command, control and communications combat assessment, methodology, simulation, games, analysis, effectiveness, measures.

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This is an interim report on the 1986 work on the extension and development of the C3EVAL model. The model is to permit assessment of the effects on combat of changes in command and control processes and communications network structure and capacity. The model has had a partial pre-processor added to assist possible analyst/model users to input data and a post-processor to provide graphic display of some outputs. The command structure includes the Central European command nodes from division to SHAPE for US forces. The nodes have had input and output limits added to permit representation of degraded operation as under attack or when the unit is moving. The corps level force allocation procedure has been improved. Some processes have been randomized. The corps operates on information different from that available at the division or combat unit, due to time delays, randomness, scenario inputs and corps controls allocation of corps support resources. The impact of changes in the C3 system can be seen in changes in weapon losses, non-arrival of close air support and messages delayed/lost as well as other operations-related elements.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS REPORT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) — 22C. OFFICE SYMBOL

DD FORM 1473, 84 MAR

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

IDA PAPER P-1978

C^3EVAL MODEL DEVELOPMENT — 1986

Volume II: Programmers' Manual

Robert F. Robinson, *Project Leader*
M. L. Roberson (Applications Research Corporation)
D. W. Roberson (Applications Research Corporation)

April 1987

INSTITUTE FOR DEFENSE ANALYSES

Contract MDA 903 84 C 0031
Task T-I6-309

PREFACE

This paper is a report on the continuing development of the C3EVAL model undertaken by the Institute for Defense Analyses in response to Task Order T-I6-309, Theater/Tactical Command, Control and Decision Process Analysis Methodologies subtask, Development and Test of C3EVAL Model, under Contract No. MDA 903 84C 0031. The work is for the Office of the Joint Chiefs of Staff/Command, Control and Communications (J-6). The purpose of the IDA program is to provide JCS/C3S with a means for the evaluation of theater/tactical command, control and communications in terms of military operations. The previous work was reported in IDA Paper P-1882, C3EVAL Model Development and Test, October 1985, UNCLASSIFIED.

This report is in two volumes. Volume I is the description of the model and its capabilities. Volume II is the programmers' manual.

CONTENTS

FIGURES

GLOSSARY

ATO	Air tasking order
CAP	Combat air patrol
CAS	Close air support
CRC	Control and Reporting Center
DDS	Dynamic data structure
DM	Dynamic memory
FMS	Forms Management System
IML	Input message list
IMT	Input message table
OMP	Output message process
TE	Table of equipment
TOT	Time on target
UAR	User Action Routine
WOC	Wing Operations Center

EXECUTIVE SUMMARY

This is the revised and updated Programmers' Manual for the C3EVAL model as of September, 1986. The revisions include the following:

1. Additions include nodes (shown as shaded) and paths for those nodes (Figure ES-1).

2. Rules have been added for logistics messages and previous rules have been altered to account for the additional nodes and the changes in message routing they imply.

3. Corp support artillery and helicopters have been added with rules for the allocation of corps support resources to the divisions under the command of the Corps Tac.

4. Automatic posture change has been added so that the combat unit (division or Armored Cavalry Regiment) responds to the situation it is in. Required rules have been added for notification of and approval by corps.

5. The pre- and post-processors have been extended.

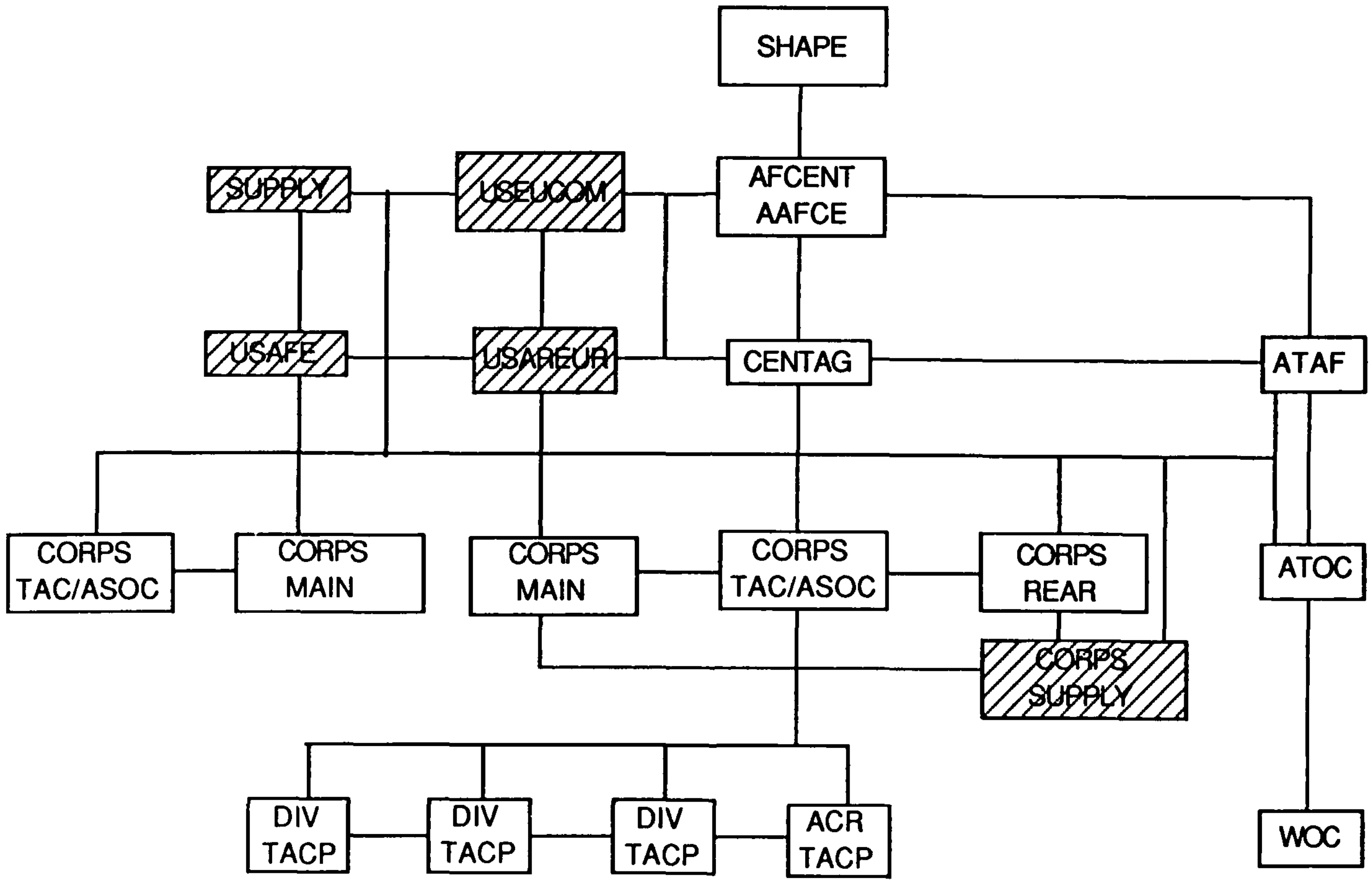


Figure ES-1. COMMAND AND CONTROL CONNECTIVITY IN MODEL

I. INTRODUCTION

This manual is to be used in conjunction with an analysts' manual for the C3EVAL model. This manual describes the C3EVAL command, control and communications model and its preprocessor and post-processor. The preprocessor consists of 62 subroutines of approximately 4,800 lines of code. The post-processor consists of 32 subroutines of approximately 2,200 lines of code. The C3EVAL (87) model has 83 subroutines and approximately 7,100 lines of code. The source code for the model is written in FORTRAN for a VAX 11/785. Each of the program structures (PREPROC, C3EVAL, POSTPROC) are presented in separate sections. There is some duplication of subroutine names in PREPROC and C3EVAL. In those cases where the subroutine code is actually used by both programs it is noted in the PREPROC description. Each description contains the data structures, notes on the program and extracts of the comments that are in the code as internal documentation. Computer listings of the codes are in the attachments.

The preprocessor utilizes the DEC Forms Management System (FMS) to communicate with the users. The post-processor is based on DECGRAPH for continuous and bar graph output. C3EVAL requires the normal FORTRAN library routines, including user options to use the random number generator.

The source code and development data files are contained in the [C3EVAL.UNCLAS] directory on the IDA VAX 11/785 computer under user identification code CAG - 060107. Some model facilities are under continuing development. Those functions that are in this status (i.e., PREPROC ground combat data, combat support data and helicopter allocation in C3EVAL) are identified in the applicable sections. Figure I-1 shows the functional and file relationships between programs. The C3EVAL input files can be modified by use of a general purpose editor and contain data preambles and comment areas to assist a user in this mode. The PREPROC work file is binary, and is not useful to a general purpose editor. All files shown may be saved for future reference and comparison of results.

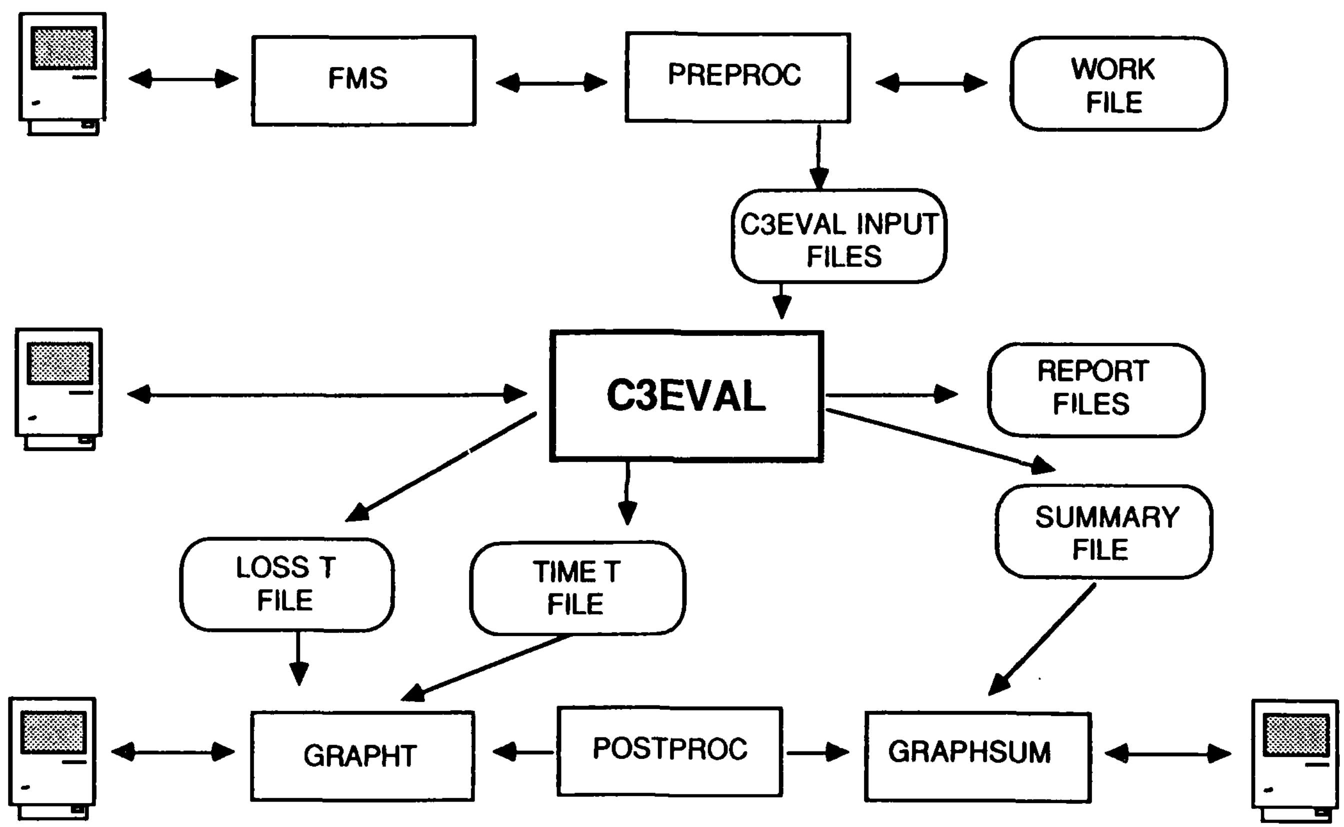


Figure I-1. C3EVAL PROGRAM FUNCTIONS

II. PROGRAM PREPROC

The preprocessor was written in order to facilitate the creation and modification of the data base required to run C3EVAL. The program is a mixture of FORTRAN subroutines and Forms Management System (FMS) commands for communication with the user. The relationships between the subroutines used by PREPROC are shown in the subroutine hierarchies in Figures II-1, II-2 and II-3. The data base for C3EVAL is 11 data sets contained in five input files.[1] The elements of data sets are inter-referenced during validation. The preprocessor also allows the user to specify names to identify data bases. The preprocessor is menu driven with scrolling in fields where it is required. This simplifies what the user needs to know in order to create the data base because the user does not need to know all the ins and outs of an editor. Whenever an invalid input is received, an error message is flashed at the bottom of the screen. Figure II-4 presents the main menu for the preprocessor as shown on the terminal. When the user indicates the EXIT function, output file dispositions are queried by the preprocessor.

A. EXIT

This option allows the user to leave the main menu and return to the main program. The main program can then save the contents of virtual memory or create the data file if the user indicates that he wants either file.

1. Subroutine FILEOUT

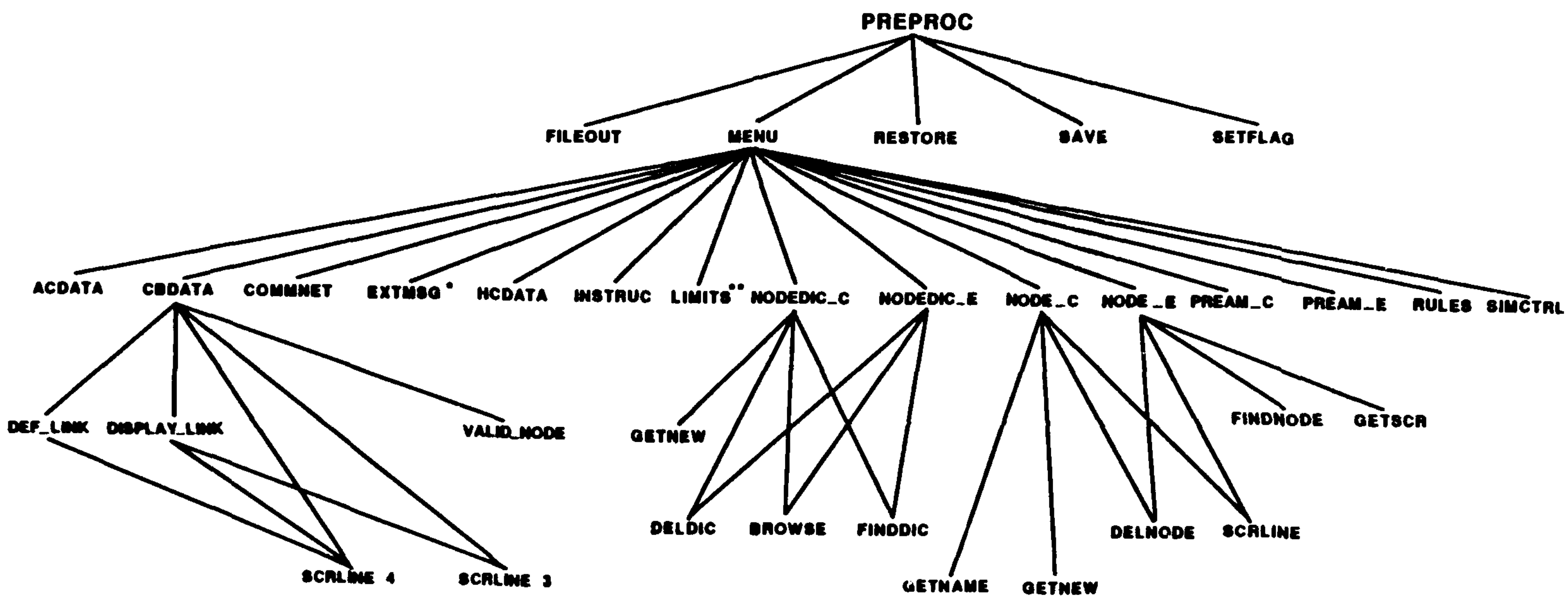
Subroutine FILEOUT creates output files to be used as data input files by program C3EVAL. File C3DATA consists of a preamble; simulation control flags; nodes and their corresponding commanders, subordinates and other communications nodes; node message input and output limits; communications networks; and external messages.

B. SUBROUTINE INSTRUC

Subroutine INSTRUC puts the instruction form on the screen and waits for the user to hit the < return > key. The instruction form contains information on how to move the cursor around

---

[1]The relationships between the user, FMS, PREPROC, C3EVAL and the POSTPROC programs and their respective data files are shown in Figure I-1 in Chapter I.



12-29-86-1M

NOTE: UTILITIES NOT SHOWN.
*SEE FIGURE II-2.
**SEE FIGURE II-3.

Figure II-1. SUBROUTINE HIERARCHY--PREPROC

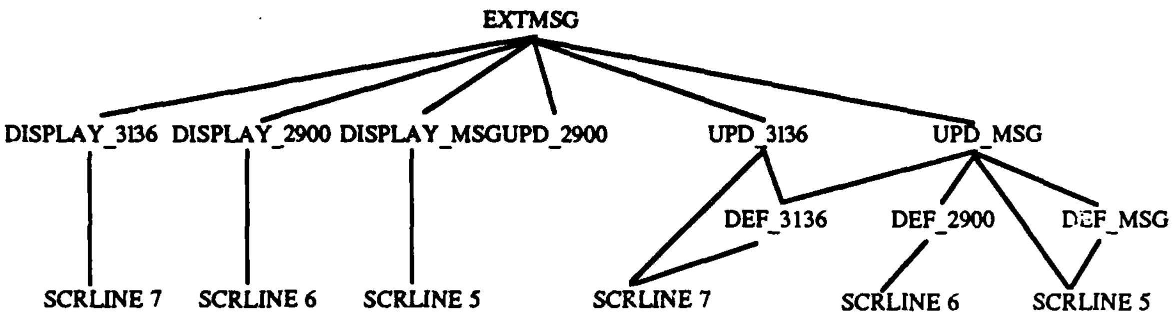


Figure II-2. SUBROUTINE HIERARCHY--EXTMSG

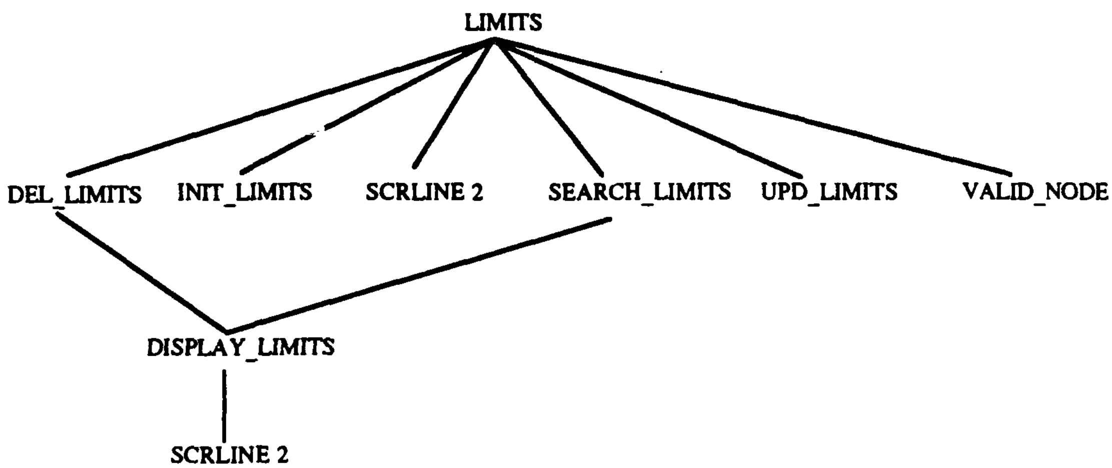


Figure II-3. SUBROUTINE HIERARCHY--LIMITS

1. EXIT	1. CREATE
2. INSTRUCTIONS	2. EDIT
3. PREAMBLE DOCUMENTATION	
4. SIMULATION CONTROL	SELECT MODE (1-2): 1
5. NODE DICTIONARY	
6. NODE	
7. LIMITS	
8. COMMUNICATIONS NETWORKS	
9. EXTERNAL MESSAGES	
10. COMBAT DATA	
11. AIRCRAFT DATA	
12. HELICOPTER DATA	
13. RULES	
SELECT OPTION NUMBER (1-12): 6	

Figure II-4. MAIN MENU FOR THE PREPROCESSOR

the screen and other special function keys (browsing up or down a queue, searching for an entry in a queue and deleting entries from a queue). Figure II-5 shows the instruction screen.

C. SUBROUTINE PREAM

This subroutine allows the user to make preamble documentation for the beginning of the data file. Each line of documentation is 60 characters long. Lines can be changed or added to the bottom, but not deleted or inserted.

D. SUBROUTINE SIMCTRL

Subroutine SIMCTRL allows the user to set the values of C3EVAL print control flags, optional output modifier times, debug output flag and debug output start and stop times. For all flags, a value of 0 represents OFF and a value of 1 represents ON. No other values are accepted as input for flags. All output times must be between 0 and 9999.

E. SUBROUTINE NODEDIC

Subroutine NODEDIC allows the user to create and edit entries within the node dictionary. This dictionary identifies the acceptable unit names and abbreviations for the scenario and associates a numeric type for each name. Create mode allows the user to create new types and

MOVING FROM ONE FIELD TO ANOTHER:
NEXT FIELD = < TAB >
PREVIOUS FIELD = < BACKSPACE >

SCROLLING IN A SCROLLED AREA:
SCROLL NEXT = DOWN_ARROW
SCROLL PREVIOUS = UP_ARROW
EXIT SCROLLED AREA NEXT = < PF1 > DOWN_ARROW
EXIT SCROLLED AREA PREVIOUS = < PF1 > UP_ARROW

SEARCHING FOR ENTRY = KEYPAD 4
NOTE: WHEN RESPONDING TO PROMT HIT < ENTER > NOT < RETURN >

DELETING ENTRY = KEYPAD 6
NOTE: WHEN RESPONDING TO PROMPT HIT < ENTER > NOT < RETURN >

LOOKING AT ENTRIES:
NEXT SCREEN = KEYPAD 2
PREVIOUS SCREEN = KEYPAD 8

HIT < RETURN > TO MAIN MENU

Figure II-5. INSTRUCTION SCREEN

names to correspond to them. Edit mode allows the user to change types, i.e., change all entries of type "100" to type "200." The user can also change or add names. Both modes allow the user to delete all entries of the current type. Only edit mode allows the user to search for a particular type. Figure II-6 is an example showing that type "300" is a division and it has acceptable abbreviations of "div" or "DIV."

1. Subroutine BROWSE

Subroutine BROWSE is used for sequentially searching through the node dictionary while in edit mode. The current contents of the screen shows all dictionary entries corresponding to the current type number. If the user hits the browse up key then the subroutine gets the locations of all entries corresponding to the previous type. If no previous type exists, the browse up key sends the "Top of Queue" message to user and keeps the pointers to the current dictionary entries. If the user hits the browse down key, then the subroutine gets the locations of all entries corresponding to the next type. If no next type exists, the browse down key sends the "Bottom of Queue" message to user and keeps the pointers to the current dictionary entries.

```
TYPE            NAME
 300          div
              DIVISION
              DIV

MODE: EDIT

                  HIT < RETURN > TO RETURN TO MAIN MENU
```

Figure II-6. DATA DICTIONARY SCREEN

2. Subroutine DELDIC

Subroutine DELDIC is used for deleting all entries in the node dictionary corresponding to the current type, i.e., all entries that are on the screen when the user hits the appropriate key. Subroutine DELDIC has two pointers, TOP and BOTTOM, which point to the first and last entries of the current type. Subroutine DELDIC starts at location TOP and walks through the queue using the pointers which are sorted by type. Each entry encountered is removed from both dictionary queues and the entry's virtual memory space is released for future use. Subroutine DELDIC stops when it encounters the memory location pointed to by BOTTOM.

3. Subroutine FINDDIC

This subroutine is called when the user hits the find key while editing the node dictionary. The type to search for is input by the user. Then Subroutine FIND is called to get the pointer to the first occurrence of the input type within the node dictionary. The pointer returned by Find is stored in TOP. Since all entries of the same type are grouped together, the dictionary is walked through starting at location TOP until the last entry of the input type is found. The pointer to the last entry is stored in BOTTOM. If the type to search for is not found in the node dictionary (Find returns a 0), then get first type that is in node dictionary (set TOP to 0).

4. Subroutine GETNEW

Subroutine GETNEW gets from the user a name that is not an already existing entry. The field to input from and the queue to search are both parameters. The offset from the beginning of a queue element to compare on is also a parameter. If the user input entry is found in the specified queue, an error message is sent and user must input again. If the user input entry is not found in the specified queue, the input is returned to the calling routine.

F. SUBROUTINE NODE

Subroutine NODE allows the user to create and edit the node data set. In create mode the user creates new nodes and gives information about the node. This information includes the main node's commander, the main node's subordinates and any other nodes that the main node can communicate with. There can also be two alternate communication nodes for each node that the main node can communicate with. In edit mode, the user is allowed to change information about existing nodes. Both modes allow the user to delete the current node from the node queue. Only edit mode allows the user to search for a specific node or to browse up or down the node queue.

1. Subroutine DELNODE

Subroutine DELNODE deletes a node from the node queue. When a node is deleted, its name is set to "DELETED" and it is resorted into the node queue. The node is not removed entirely from the node queue because all other nodes that have the deleted node listed as a commander, subordinate, etc. would either find the wrong node or garbage when it accessed the pointer to the deleted node. When the node is deleted, the pointers to the commander and its alternates are set to 0. All entries in the subordinate and other network node queues are removed from the queues and their virtual memory space returned for future use.

2. Subroutine FINDNODE

This subroutine is called when the user hits the find key while editing a node. The user inputs the name of the node to search for. Then Subroutine FIND is used to get the pointer to the entry in the node queue that has the input name. If the node name is found in the node queue, the current position is set to the pointer to the entry. If the node name is not found (find returns a 0), then the current position is set to 0 and a message is sent to the user.

3. Subroutine GETNAME

Subroutine GETNAME is used to get a valid node name from the user. If the input name is an already existing node, the pointer to that node is returned. Otherwise, if the name contains an entry in the node dictionary, a new node is created with its name being the input name. This new node is sorted into the node queue and the pointer to its location is the value returned by GETNAME. If the input name is neither an already existing node nor a node name that contains an

entry in the node dictionary, the input name is illegal. The subroutine sends an error message to the user and gets another node name from the user.

4. Subroutine GETNEW

Subroutine GETNEW gets from the user a name that is not an already existing entry. The field to input from and the queue to search are both parameters. The offset from the beginning of a queue element to compare on is also a parameter. If the user input entry is found in the specified queue, an error message is sent and user must input again. If the user input entry is not found in the specified queue, the input is returned to the calling routine.

5. Subroutine GETSCR

Subroutine GETSCR gets from the user a sequence of valid node names from a scrolled area. A valid node name is either a node already in the queue or a node name that contains a word that is in the node dictionary. If each node name in the sequence already exists, the pointer to its location in virtual memory is saved. If the node contains a word that is in the dictionary, a new entry for the queue is created and its pointer is saved. Otherwise the node name is illegal and the user must input another node name. The values returned are the three pointers to the nodes input by the user. The three nodes are a main node and its two alternate nodes.

6. Subroutine SCRLINE

Subroutine SCRLINE is used to create a string to output to the scrolled area for node queue. The scrolled line contains six fields:

- Main node name,
- Main node id,
- First alternate's name,
- First alternate's id,
- Second alternate's name, and
- Second alternate's id.

Each name is a string of 12 characters and each id is a string of 4 characters. Each name and id is found by using the appropriate pointer to get the node's location in memory and picking up the name and id from the appropriate offsets. When the resulting string is passed to FMS, it will be

parsed and each value will be sent to the corresponding field. If the pointer is null, the values are set to their defaults.

G. SUBROUTINE LIMITS

Subroutine LIMITS creates and edits the entries in the node message input and output limits queue. Default entries are inserted when the user first makes the queue. One default entry is created for each node in the node queue. The user can add, change or delete any entries in the queue. Searching for specific entries is allowed for ease of editing the queue and is initiated by hitting the numeric keypad 4 key. Deleting an entry is performed by hitting the numeric keypad 6 key.

1. Subroutine DEL_LIMITS

Subroutine DEL_LIMITS deletes the current entry from the node message input and output queue. The subroutine double checks to insure that the user really wants to delete the current entry. The updated queue is displayed in the scrolled area after the entry is removed from the queue and its memory space is released for future use.

2. Subroutine DISPLAY_LIMITS

Subroutine DISPLAY_LIMITS displays a portion of the node message input and output limits queue in the scrolled area. The first entry displayed is pointed to by the parameter Pos. The other scrolling control parameters are initialized and returned to the calling procedure to facilitate the scrolling functions to allow viewing of the portions of the queue which are not currently in the scrolled area. Subroutine SCRLINE2 is used to create each line to output to the scrolled area.

3. Subroutine INIT_LIMITS

Subroutine INIT_LIMITS creates a default entry in the node message input and output limit queue for each entry in the node queue. The default values are only created when the user first creates the queue, i.e., if a node is added to the node queue after the limits queue is made, a default entry will not be made in the limits queue. The default queue values for time is 0, input and output limits are 99, random distribution for capacity is 1 and random distribution for delete and CAS are 0.

4. Subroutine SCRLINE2

Subroutine SCRLINE2 is used to create a string to output to the scrolled area for node message input and output limits. The scrolled line contains 11 fields consisting of:

- Node name
- Node number,
- Node type,
- Time,
- Input hold,
- Kill limits,
- Output hold,
- Kill limits,
- Random distributions for capacity,
- Delete, and
- CAS.

For additional detail, see Subroutine SCRLINE.

5. Subroutine SEARCH_LIMITS

Subroutine SEARCH_LIMITS allows the user to search for an entry in the node message input and output limits queue. There are five ways to search for an entry. The first three ways search the queue starting at the current entry, searching down the queue to the bottom, and search on either node name, node number or time. The other two ways to search start at the top of the queue and work down to the bottom. They use two keys and search on either node name and time or node number and time. All five methods only look for the first entry which satisfies the conditions.

6. Subroutine UPD_LIMITS

Subroutine UPD_LIMITS retrieves the values from the current line of the node message input and output limits scrolled area. The parameter PLimits is used as a pointer into dynamic memory to update the values of the current entry of the limits queue. Only the time, input and output limits and random distributions are updated. The values for the node's identification are not updated here because they require verification.

7. Subroutine VALID_NODE

Subroutine VALID_NODE determines if a unit identifier is valid. If user gave the unit name only, then the name must exist in the node queue. If user gave the unit number only, the number must exist in the node queue. If user gave the unit number and name, then the node must exist in the node queue and the name and number must match. The subroutine returns a pointer to the node if it is valid, otherwise it returns a null pointer.

H. SUBROUTINE COMMNET

Subroutine COMMNET creates and edits the entries in the communications link dictionary and communications link queue. The communications link dictionary contains the valid link types and their corresponding capacities and names. The communications link queue contains all links between all nodes. Each entry in the link queue contains:

- Time to change link,
- The two nodes that the link connects,
- The link type and capacity, and
- The random distribution type for a lost message.

1. Subroutine DEF_LINK

Subroutine DEF_LINK creates an entry in the communications link queue with default values. The entry is added to the bottom of the queue. The default values are obtained by calling Subroutine SCRLINE4 with a null pointer.

2. Subroutine DISPLAY_LINKS

Subroutine DISPLAY_LINKS displays the communications link dictionary and the communications link queue in their scrolled areas. The first entries to be displayed in the link dictionary and link queue scrolled areas are pointed to by parameters Pos1 and Pos2, respectively. The other parameters are initialized and returned to the calling procedure to facilitate the scrolling functions to allow viewing of the portions of the queues which are not currently in the scrolled areas. Subroutines SCRLINE3 and SCRLINE4 are used to create each line to output to the scrolled areas for the link dictionary and link queue, respectively.

3. Subroutine SCRLINE3

Subroutine SCRLINE3 is used to create a string to output to the scrolled area for communications link dictionary. The scrolled line contains three fields consisting of:

- Link type,
- Link capacity, and
- Link name.

4. Subroutine SCRLINE4

Subroutine SCRLINE4 is used to create a string to output to the scrolled area for communications link queue. The scrolled line contains six fields consisting of:

- Time to change link,
- The two nodes the link connects,
- The link type,
- The link capacity, and
- The random distribution type for a lost message.

For additional details, see Subroutine SCRLINE.

5. Subroutine VALID_NODE

Subroutine VALID_NODE determines if a unit identifier is valid. If user gave the unit name only, then name must exist in the node queue. If user gave the unit number only, the number must exist in the node queue. If user gave the unit number and name, then the node must exist in the node queue and the name and number must match. The subroutine returns a pointer to the node if it is valid, otherwise it returns a null pointer.

I. SUBROUTINE EXTMSG

Subroutine EXTMSG allows the user to create and edit the entries in the external message queue, including the additional data lines for any message that requires additional data. There are two types of external messages that require additional data. A "2900" is a preplanned CAS message and requires one additional data line. A "3136" is an Intel Report message and can have one or more additional data lines.

1. Subroutine DEF_MSG

Subroutine DEF_MSG creates an entry in the external message queue. The entry is added to the bottom of the queue with all values being set to their default values. If the message requires additional data, the appropriate subroutine is called to create a default entry for the additional data line.

2. Subroutine DEF_2900

Subroutine DEF_2900 creates an additional data line for a message of type 2900. Values are set to their defaults and entry is linked with its corresponding external message.

3. Subroutine DEF_3136

Subroutine DEF_3136 creates an entry in the additional data line queue for a message of type 3136. Values are set to their defaults and the entry is added to the bottom of the queue for its corresponding external message.

4. Subroutine DISPLAY_MSG

Subroutine DISPLAY_MSG displays messages from the external message queue in the scrolled area. The first entry displayed is pointed to by the parameter Pos. The other parameters are initialized and returned to the calling procedure to facilitate the scrolling functions to allow viewing of the messages on the queue which are not currently in the scrolled area. Subroutine SCRLINE5 is used to create each line to output to the scrolled area.

5. Subroutine DISPLAY_2900

When a message of type 2900 is displayed by DISPLAY_MSG, Subroutine DISPLAY_2900 displays the additional data line in the optional portion of the screen. Subroutine SCRLINE6 is used to create the line to output to the scrolled area.

6. Subroutine DISPLAY_3136

When a message of type 3136 is displayed by DISPLAY_MSG, Subroutine DISPLAY_3136 displays a portion of the additional message queue in the optional portion of the screen. The first entry displayed is pointed to by the parameter Pos. The other parameters are

initialized and returned to the calling procedure to facilitate the scrolling functions to allow viewing of the portions of the queue which are not currently in the scrolled area. Subroutine SCRLINE7 is used to create each line to output to the scrolled area.

7. Subroutine SCRLINE5

Subroutine SCRLINE5 is used to create the line to output to the scrolled area for external messages. The line contains eight fields consisting of:

- Send Time,
- Message type,
- Originating unit type,
- Destination unit id,
- Message create time,
- Print flag,
- Priority, and
- Maximum network time.

For additional details, see Subroutine SCRLINE.

8. Subroutine SCRLINE6

Subroutine SCRLINE6 is used to create the line to output for an additional data line for an external message of type 2900. The line contains five fields consisting of:

- Support unit id,
- Earliest support time,
- Latest support time,
- Aircraft type, and
- Number of aircraft.

9. Subroutine SCRLINE 7

Subroutine SCRLINE7 is used to create the line to output to the scrolled area for an additional data line for an external message of type 3136. The line contains five fields consisting of:

- Unit id,
- Report time,

- Foe's type,
- Posture, and
- Id.

For additional details, see Subroutine SCRLINE.

10. Subroutine UPD_MSG

Subroutine UPD_MSG processes any changes the user makes to a field which is in the form for an external message. If any changes were made, the appropriate validation is made and if the new value is valid, then internal data is updated to reflect the new value. UPD_MSG processes the field terminator and sets parameter Done to one if field terminator was a < return >. If field terminator was an"illegal exit scrolled area forward" and the current entry in the external message queue contains additional data, then UPD_MSG sets current workspace to the form containing the additional data area to allow user to make changes to the additional data.

11. Subroutine UPD_2900

Subroutine UPD_2900 processes any changes the user makes to a field which is in the form for additional data line for an external message of type 2900. If any changes were made, the appropriate validation is made. If the new value is valid, then the internal data is updated to reflect the new value. UPD_2900 processes the field terminator and sets parameter Done to one if field terminator was a < return>. If field terminator was"illegal," UPD_2900 sets current workspace to the form containing the external messages.

12. Subroutine UPD_3136

Subroutine UPD_3136 is similar to Subroutine UPD_2900, except it processes for messages of type 3136 instead of type 2900.

J. SUBROUTINE CBDATA

Subroutine CBDATA is not implemented yet.

K. SUBROUTINE ACDATA

Subroutine ACDATA is not implemented yet.

L. SUBROUTINE HCDATA

Subroutine HCDATA is not implemented yet.

M. SUBROUTINE RULES

Subroutine RULES is not implemented yet.

N. UTILITIES

1. Subroutine CHRINT

Subroutine CHRINT converts a character string representation of an integer to its integer equivalent. The length of the string is specified by parameter ISize.

2. Subroutine DMINIT

Same as Subroutine DMINIT in Chapter III, Program C3EVAL, Section A.2.

3. Subroutine FIND

Same as Subroutine FIND in Chapter III, Program C3EVAL, Section E.1.

4. Subroutine FIND_INDIRECT

Subroutine FIND_INDIRECT finds a pointer in a queue where the value to be searched for is not stored in the queue. The value to be searched for is indirectly pointed to by PIN+2, where PIN is a pointer to an entry in the queue. The parameter N is the offset from the location pointed to by PIN+2 where the value to be searched for is located. The subroutine returns POUT, which is the pointer to the entry in the queue which indirectly points to the value which matches parameter ID.

5. Subroutine GETTYP

Subroutine GETTYP searches the dictionary for each word that occurs in the node name. If an occurrence of a word in node name is found, it then returns the type corresponding to the dictionary entry--otherwise returns a string of blanks.

6. Subroutine GETWORD

Subroutine GETWORD finds the first word that is contained in a string. If the string passed in is blank, GETWORD returns blanks for the string and the word--otherwise the first word within the string is found and saved in IWORD. Then the word is removed from the string and GETWORD returns the resulting string and IWORD.

7. Subroutine GIMME

Same as Subroutine GIMME in Chapter III, Program C3EVAL, Section E.2.

8. Subroutine INTCHR

Subroutine INTCHR converts an integer to its ASCII representation. The parameter ISize is the number of digits to convert. Note that the maximum length of the string is 12 characters by declaration.

9. Subroutine POUT

Same as Subroutine POUT in Chapter III, Program C3EVAL, Section E.3.

10. Subroutine RANGE

Subroutine RANGE determines if a value is within the specified range. The minimum and maximum values, along with the character string representation of the number to validate, are parameters. The subroutine returns a 1 if the value is within the range and a 0 if the value is not within the range.

11. Subroutine RELEASE

Same as Subroutine RELEASE in Chapter III, Progam C3EVAL, Section E.5.

12. Subroutine RESTORE

Same as Subroutine RESTORE in Chapter III, Program C3EVAL, Section E.6.

13. Subroutine RETSC

Subroutine RETSC returns the values that are currently stored in consecutive fields. The fields to return the values are delineated by the parameters start and finish. Start is the field number of the first field to return a value. Finish is the field number of the last field to return a value. The subroutine returns a string which is the concatenation of each of the strings containing the value for each of the fields.

14. Subroutine SAVE

Same as Subroutine SAVE in Chapter III, Program C3EVAL, Section E.7.

15. Subroutine SCRBK

Subroutine SCRBK is called whenever the user hits the up_arrow key while in a scrolled area. If the current line of the scrolled area is not the top line, the new current line becomes the line above the current line. If the current line of the scrolled area is the top line and there are undisplayed lines above the current line, each line of the scrolled area is moved and and the new line is displayed at the top of the scrolled area.

16. Subroutine SCRFWD

Subroutine SCRFWD is called whenever the user hits the down_arrow key while in a scrolled area. If the current line of the scrolled area is not the bottom line, the new current line becomes the line below the current line. If the current line of the scrolled area is the bottom line and there are undisplayed lines below the current line, each line of the scrolled area is moved up and the new line is displayed at the bottom of the scrolled area.

17. Subroutine SETFLAG

Subroutine SETFLAG creates the data structure that contains the print control flags, optional print modifiers, debug print flag and debug print start and stop times. All values are initialized to 0. Since all flags are only one character, but are stored in four character fields, the first character of the field is initialized to 0. However, since the optional print modifiers and debug print start and stop times are four character modifiers and debug print start and stop times are four characters fields that are right justified, the 0 is the last location.

18. Subroutine SNAPQ

Subroutine SNAPQ inserts an entire queue of records into another queue or records. It assumes that all records in the queue being added have the same value being sorted on and therefore can be inserted as one large record. Note: A queue of one record can be inserted by passing the same pointer for both the top and bottom of the queue to be added. It assumes that there is a corresponding back pointer for the forward pointer. It assumes back pointer is offset from its forward pointer by one.

19. Subroutine UNSNAP

Subroutine UNSNAP removes an entry from a queue. It assumes that there is a corresponding back pointer for each forward pointer. It assumes back pointer is offset from its forward pointer by one. It sets forward pointer of previous node to next node and sets back pointer of next node to previous node.

20. Function VALID1

Function VALID1 is a Field Completion User Action Routine. VALID1 checks to see that the inputted value is between one and a maximum value. The maximum value is stored in Named Data, which is an FMS data structure.

O. DATA STRUCTURES

The data dictionary has its root in COMMON/LOCATE/MROOT. It is linked together by pointers based in the data block MROOT. The subsequent data blocks are created by calls to Subroutine GIMME which acquires data blocks from dynamic memory. The length of each block is a fixed number in code and is shown on the following forms as Block Size. This section lists the data blocks; defines their elements; gives the type of each variable; identifies the routine that creates the block and the ones that delete the block, if applicable; and specifies the location of the root. The symbols used in the DDSs documented in this section are P, pointer to another DDS, and C, character with length given in number of characters.

Block Name: MROOT Block Size: 30
Use: Contains all root pointers for virtual memory.

Created by: PREPROC
Deleted by: Not applicable
Root: COMMON/LOCATE/MROOT Date: 31 Dec 86

Index	Element name	Type	Element meaning/use
1	PDICN1	P	NODE dictionary (sorted by TYPE)
2	PDICN2	P	NODE dictionary (sorted by NAME)
3	PNODE	P	NODE queue
4	PREAMB	P	Preamble documentation
5	PFLAG	P	Print control flags
6	PLIMIT	P	LIMITS queue
7	PDICL	P	LINK dictionary
8	PLINK	P	Communications LINK queue
9	PMSG	P	External MESSAGES queue

Block Name:	PDICN	Block Size:	6
Use:	Dictionary of all valid node types. Each type number has one or more unit names that correspond to that type.		
Created by:	NODEDIC		
Deleted by:	DELDIC		
Root:	MROOT(1)	Date:	31 Dec 86

Index	Element name	Type	Element meaning/use
1	PDICN1	P	Next dictionary entry (by TYPE)
2	PREV1	P	Previous dictionary entry (by TYPE)
3	PDICN2	P	Next dictionary entry (by NAME)
4	PREV2	P	Previous dictionary entry (by NAME)
5	NAME	C*12	Dictionary entry
6	TYPE	C*4	Unit type of dictionary entry

Block Name: PNODE Block Size: 10

Use: Queue containing all nodes for the scenario. Each entry in the queue also has all communication paths that pertain to the node.

Created by: GETNAME,GETSCR,NODE

Deleted by: DELNODE

Root: MROOT(3) Date: 31 Dec 86

Index	Element name	Type	Element meaning/use
1	PNODEF	P	Next NODE
2	PNODEB	P	Previous NODE
3	NAME	C*12	NODE name
4	UNIT	C*4	NODE number
5	TYPE	C*4	NODE type
6	PCMDR	P	NODE's commander
7	PCMDR1	P	1st alternate for commander
8	PCMDR2	P	2nd alternate for commander
9	PSUBQ	P	NODE's subordinate queue
10	PCOMQ	P	NODE's network queue

Block Name:	PSUBQ
Use:	Queue of all subordinate communications paths for a specified node. Each entry in the queue can also have two alternate communications paths for the subordinate.
Created by:	NODE
Deleted by:	DELNODE
Root:	PNODE(9)

Block Size: 5

Date:31 Dec 86

Index	Element name	Type	Element meaning/use
1	PSUBQF	P	Next subordinate
2	PSUBQB	P	Previous subordinate
3	PSUB	P	NODE's subordinate
4	PSUB1	P	1st alternate for subordinate
5	PSUB2	P	2nd alternate for subordinate

Block Name: PCOMQ Block Size: 5

Use: Queue of all other communications paths for a specified node. Each entry in the queue can also have two alternate communications paths.

Created by: NODE

Deleted by: DELNODE

Root: PNODE(10) Date: 31 Dec 86

Index	Element name	Type	Element meaning/use
1	PCOMQF	P	Next network NODE
2	PCOMQB	P	Previous network NODE
3	PCOM	P	Network NODE
4	PCOM1	P	1st alternate for network NODE
5	PCOM2	P	2nd alternate for network NODE

Block Name:	PREAMB	Block Size:	7
Use:	Linked list of preamble documentation line. Each line is 60 characters long. Each line is subdivided into 5 parts for storage.		
Created by:	PREAM		
Deleted by:	Not applicable		
Root:	MROOT(4)	Date:	31 Dec 86

Index	Element name	Type	Element meaning/use
1	PREAMBF	P	Pointer to next line
2	PREAMBB	P	Pointer to previous line
3-7	LINE	C*60	Line of documentation

Block Name:	PFLAG	Block Size: 26
Use:	List of all print control flags, optional output modifier times, debug output flag and debug output start and stop times. Note (for all flags): 0-->OFF, 1-->ON.	
Created by:	SETFLAG	
Deleted by:	Not applicable	
Root:	MROOT(5)	Date: 31 Dec 86

Index	Element name	Type	Element meaning/use
1	FLAG1	C*1	All messages at alternate dest.
2	FLAG2	C*1	All messages on input queues
3	FLAG3	C*1	All messages on output queues
4	FLAG4	C*1	All messages on future queues
5	FLAG5	C*1	All messages being held
6	FLAG6	C*1	All messages being deleted
7	FLAG7	C*1	Status of rule structure
8	FLAG8	C*1	CAS take off scheduled
9	FLAG9	C*1	Not assigned
10	FLAG10	C*1	Not assigned
11	FLAG11	C*1	Tracked messages at alternate dest.
12	FLAG12	C*1	Tracked messages on input queues
13	FLAG13	C*1	Tracked messages on ouput queues
14	FLAG14	C*1	Time T output on file 14 required
15	FLAG15	C*1	Combat loss vector
16	FLAG16	C*1	Force ratio calculations
17	FLAG17	C*1	Rule status at final time
18	FLAG18	C*1	Not assigned
19	FLAG19	C*1	Random processing required
20	FLAG20	C*1	Used internally for sum of flags
21	MOD1	C*4	Optional output restricted to this node
22	MOD2	C*4	Optional output starts at this time
23	MOD3	C*4	Optional output stops at this time
24	DEBUG1	C*1	Debug output flag
25	DEBUG2	C*4	Debug output start time
26	DEBUG3	C*4	Debug output stop time

Block Name:	PLIMIT	Block Size:	11
Use:	Queue of all node message input and output limits.		
Created by:	INIT_LIMITS,LIMITS		
Deleted by:	DEL_LIMITS		
Root:	MROOT(6)	Date:	31 Dec 86

Index	Element name	Type	Element meaning/use
1	PLIMITF	P	Next LIMIT
2	PLIMITB	P	Previous LIMIT
3	PNODE	P	Corresponding NODE
4	TIME	C*4	Time to change limits
5	LMN1	C*3	Input limit - Hold
6	LMN2	C*3	Input limit - Kill
7	LMO1	C*3	Output limit - Hold
8	LMO2	C*3	Output limit - Kill
9	MLEN	C*2	Random distribution - Capacity
10	MDEL	C*2	Random distribution - Delay
11	CASD	C*2	Random distribution - CAS delay

Block Name: PDICL | Block Size: 5

Use: Dictionary of communication link types and their corresponding capacities and names.

Created by: COMMNET

Deleted by: Not applicable

Root: MROOT(7) | Date: 31 Dec 86

Index	Element name	Type	Element meaning/use
1	PDICLF	P	Next dictionary entry
2	PDICLB	P	Previous dictionary entry
3	TYPE	C*4	LINK type
4	LENGTH	C*8	LINK capacity
5	NAME	C*12	LINK name

Block Name: PLINK

Block Size: 8

Use: Queue of all communications links between all nodes.

Created by: DEF_LINK

Deleted by: Not applicable

Root: MROOT(8)

Date: 31 Dec 86

Index	Element name	Type	Element meaning/use
1	PLINKF	P	Next LINK
2	PLINKB	P	Previous LINK
3	TIME	C*4	Time to change LINK
4	PNODE1	P	Node
5	PNODE2	P	Node
6	TYPE	C*4	LINK type
7	LENGTH	C*8	LINK capacity
8	RANDDIST	C*2	Random distribution type for lost message

Block Name: PMSG — Block Size: 11
Use: Queue of all external messages.

Created by: DEF_MSG
Deleted by: Not applicable
Root: MROOT(9) — Date: 31 Dec 86

Index	Element name	Type	Element meaning/use
1	PMSGF	P	Next EXTERNAL MESSAGE
2	PMSGB	P	Previous EXTERNAL MESSAGE
3	SEND	C*4	Send time
4	TYPE	C*4	Message type
5	ORIG	C*4	Originating unit type
6	DEST	C*4	Destination unit id
7	CREATE	C*4	Message create time
8	PRINT	C*1	Print flag
9	PRIOR	C*2	Priority
10	MAXNET	C*4	Maximum network time
11	ADATA	P	*Additional data*

Block Name: P2900 Block Size: 7

Use: Single entry for additional data for an external message of type 2900 (preplanned CAS).

Created by: DEF_2900

Deleted by: Not applicable

Root: PMSG(11) Date: 31 Dec 86

Index	Element name	Type	Element meaning/use
1	P2900F	P	Next 2900 data line
2	P2900B	P	Previous 2900 data line
3	UNITID	C*4	Support unit id
4	EARLY	C*4	Earliest support time
5	LATE	C*4	Latest support time
6	TYPE	C*1	Aircraft type
7	NUMBER	C*4	Number of aircraft

Block Name: P3136 Block Size: 7

Use: Queue of additional data lines for an external message of type 3136 (Intel Report).

Created by: DEF_3136
Deleted by: Not applicable
Root: PMSG(11) Date: 31 Dec 86

Index	Element name	Type	Element meaning/use
1	P3136F	P	Next 3136 data line
2	P3136B	P	Previous 3136 data line
3	UNITID	C*4	Unit id
4	TIME	C*4	Report time
5	TYPE	C*4	Foe type
6	POSTURE	C*1	Foe's posture
7	FOEID	C*4	Foe's id

P. PROGRAM NOTES

Most of the interfacing with the screen is accomplished by using FMS provided routines and structures. One tool provided by FMS is a field completion User Action Routine (UAR), a function called by FMS when the user completes his entry for a field. The function can process the value that the user input to determine if it is a legal entry. The function returns a value which tells FMS to either accept the user input or to get another value from the user. Named Data is another tool provided by FMS. These are data values that have names so that they can be accessed by an FMS command. In this way, values can be associated with particular fields but not hard-wired into the actual code. More detailed information on UARs, Named Data or any of the FMS provided routines can be obtained from VAX manuals on FMS.

Q. INTERNAL CODE DOCUMENTATION

```
PROGRAM PREPROC

PURPOSE:  CREATE DATA FILE TO BE USED AS INPUT FOR
   PROGRAM C3EVAL

INITIALIZE FMS
IF WORKFILE EXISTS LOAD DYNAMIC MEMORY ELSE INITIALIZE
 DYNAMIC MEMORY
PROCESS ALL MENU REQUESTS
SAVE CONTENTS OF MEMORY AND COMMON'S
CREATE OUTPUT FILE
CLEAN UP
```

SUBROUTINE ACDATA
SUBROUTINE IS NOT IMPLEMENTED YET

```
SUBROUTINE BROWSE(NWORD, FLAG)

PURPOSE:  USED WHEN EDITING NODE DICTIONARY. MOVES UP OR DOWN
  SCREEN, I.E. GETS ENTRIES CORRESPONDING TO PREVIOUS OR NEXT

PARAMETERS:
   NWORD    ==> OFFSET TO COMPARE ON
   FLAG     ==> DIRECTIONAL FLAG
                0 -> SEARCH DOWN
                1 -> SEARCH UP

IF NO NEXT ENTRY THEN SEND APPROPRIATE PROMPT
SET TOP TO NEXT ENTRY
FIND ALL ENTRIES OF SAME TYPE AS TOP
SET BOTTOM TO LAST ENTRY OF SAME TYPE AS TOP

IF SEARCHING UP THEN SWITCH TOP AND BOTTOM POINTERS AND
 SUBTRACT ONE TO GET FORWARD POINTERS INSTEAD OF PREVIOUS
 POINTERS.
```

```
SUBROUTINE CBDATA
SUBROUTINE IS NOT IMPLEMENTED YET
```

```
SUBROUTINE CHRINT(STRING, ISIZE, VALUE)

SUBROUTINE CONVERTS CHARACTER STRING REPRESENTATION OF
AN INTEGER INTO IT'S INTEGER VALUE

PARAMETERS:
   STRING    ==>   STRING TO CONVERT TO INTEGER EQUIVALENT
   ISIZE     ==>   NUMBER OF CHARACTERS TO CONVERT
   VALUE     ==>   INTEGER RESULT OF CONVERSION
```

SUBROUTINE COMMNET

PURPOSE: USED TO CREATE AND EDIT THE COMMUNICATION LINK DICTIONARY AND COMMUNICATION LINK DATA SET

FMS TERMINATOR CODES

CONSTANT
MAXLINE1 = NUMBER OF SCROLLED LINES IN THE LINK DICTIONARY
MAXLINE2 = NUMBER OF SCROLLED LINES IN THE LINK QUEUE AREA

INITIALIZE SCREEN AND POINTERS

GET CURRENT VALUES AND THEN GET NEW VALUE FROM USER

IF NO CHANGE TO FIELD THEN PROCESS FIELD TERMINATOR

ADDING NEW ENTRY TO DICTIONARY
VALIDATE THAT LINK TYPE DOESN'T ALREADY EXIST

SAVE NEW ENTRY

USE SCRLINE3 TO GET DEFAULT VALUES

SET FLDNAM TO LINK CAPACITY & IGNORE FIELD TERMINATOR TO FORCE USER TO GIVE LINK CAPACITY

CHANGING EXISTING TYPE IN DICTIONARY
VALIDATE THAT LINK TYPE DOESN'T ALREADY EXIST

ILLEGAL ENTRY, MUST GIVE VALID TYPE FIRST

CHANGE EXISTING CAPACITY IN DICTIONARY

ILLEGAL ENTRY, MUST GIVE VALID TYPE FIRST

CHANGE EXISTING NAME IN DICTIONARY
ADD ENTRY TO LINK QUEUE

CHANGE EXISTING TIME

IF NULL ENTRY THEN BRANCH TO PROCESS FIELD TERMINATOR

ADD ENTRY TO LINK QUEUE

CHANGE NODE IDENTIFIER

ADDING NEW ENTRY TO LINK QUEUE
VALIDATE THAT LINK TYPE EXISTS

CHANGE EXISTING LINK TYPE FOR AN ENTRY IN LINK QUEUE
VALIDATE THAT LINK TYPE EXISTS

ILLEGAL - MUST GIVE LINK TYPE BEFORE LINK CAPACITY

CHANGING EXISTING LINK CAPACITY

ADDING NEW ENTRY TO LINK QUEUE

CHANGE EXISTING RANDOM DISTRIBUTION VALUE

PROCESS FIELD TERMINATOR

IF USER INPUTTED A NEW UNIT NUMBER THEN SKIP OVER
UNIT NAME FIELD.
IF USER INPUTTED A NEW UNIT NAME THEN SKIP OVER
UNIT NUMBER FIELD.

```
SUBROUTINE DEF_LINK(BOTLINE2,MAXLINE2,PLINK)

PURPOSE: CREATES AN ENTRY IN THE LINK QUEUE WITH DEFAULT VALUE

PARAMETERS:
  BOTLINE2     ==>  THE LINE NUMBER OF THE LAST LINE DISPLAYED IN
                    THE SCROLLED AREA
  MAXLINE2     ==>  THE MAXIMUM NUMBER OF LINES THAT CAN BE
                    DISPLAYED IN THE SCROLLED AREA
  (OUTPUT)
  PLINK        ==>  POINTER TO THE ENTRY CREATED
```

```
SUBROUTINE DEF_MSG(BOTLINE1,MAXLINE1,PMSG)

PURPOSE: CREATES AN ENTRY IN THE EXTERNAL MESSAGE QUEUE
  WITH DEFAULT VALUES

PARAMETERS:
  BOTLINE1    ==> THE LINE NUMBER OF THE LAST LINE DISPLAYED IN
                  THE SCROLLED AREA
  MAXLINE1    ==> THE MAXIMUM NUMBER OF LINES THAT CAN BE
                  DISPLAYED IN THE SCROLLED AREA
  (OUTPUT)
  PMSG        ==> POINTER TO THE ENTRY CREATED

INITIALIZE DEFAULT VALUES
```

```
SUBROUTINE DEF_2900(PMSG)

PURPOSE: CREATES ADDITIONAL DATA LINE FOR AN EXTERNAL MESSAGE
  OF TYPE 2900.  INITIALIZES VALUES TO THEIR DEFAULTS.

PARAMETERS:
  BOTLINE2    ==> THE LINE NUMBER OF THE LAST LINE DISPLAYED IN
                  THE SCROLLED AREA
  MAXLINE2    ==> THE MAXIMUM NUMBER OF LINES THAT CAN BE
                  DISPLAYED IN THE SCROLLED AREA
  (OUTPUT)
  PLINK       ==> POINTER TO THE ENTRY CREATED


INITIALIZE DEFAULT VALUES
```

```
SUBROUTINE DEF_3136(BOTLINE2,MAXLINE2,PMSG,P3136)

PURPOSE: CREATES AN ENTRY IN THE QUEUE FOR ADDITIONAL DATA LIN
  FOR AN EXTERNAL MESSAGE OF TYPE 3136.  INITIALIZES VALUES TO
  THEIR DEFAULTS.

PARAMETERS:
  BOTLINE2     ==> THE LINE NUMBER OF THE LAST LINF DISPLAYED IN
                   THE SCROLLED AREA
  MAXLINE2     ==> THE MAXIMUM NUMBER OF LINES THAT CAN BE
                   DISPLAYED IN THE SCROLLED AREA
  PMSG         ==> POINTER TO THE EXTERNAL MESSAGE TO CREATE THE
                   ADDITIONAL DATA LINE FOR
  (OUTPUT)
  P3136        ==> POINTER TO THE ENTRY CREATED

INITIALIZE DEFAULT VALUES
```

SUBROUTINE DELDIC

PURPOSE: DELETE ALL ENTRIES IN NODE DICTIONARY CORRESPONDING CURRENT TYPE, I.E. ALL ENTRIES ON SCREEN.

TOP AND BOTTOM ARE POINTERS TO THE FIRST AND LAST ENTRIES OF THE CURRENT TYPE.
DO FOR ALL ENTRIES BETWEEN TOP AND BOTTOM
UNSNAP CURRENT ENTRY FROM DICTIONARY QUEUE BY TYPE
UNSNAP CURRENT ENTRY FROM DICTIONARY QUEUE BY NAME
RETURN VIRTUAL MEMORY SPACE FOR FURTHER USE

```
SUBROUTINE DEL_LIMITS(LINE1,BOTLINE1,MAXLINE1)

PURPOSE:  DELETE THE CURRENT ENTRY FROM THE NODE MESSAGE
  INPUT AND OUTPUT LIMITS QUEUE.

PARAMETERS:
   LINE1     ==> LINE NUMBER OF THE CURRENT LINE OF THE
                 SCROLLED AREA.
   BOTLINE1 ==> LINE NUMBER OF THE LAST NODE THAT IS DISPLAYED
                 IN THE SCROLLED AREA.
   MAXLINE1 ==> MAXIMUM NUMBER OF LINES THAT CAN BE DISPLAYED
                 IN THE SCROLLED AREA AT ONE TIME.


VERIFY THERE IS A NODE TO DELETE


VERIFY THAT USER REALLY WANTS TO DELETE CURRENT ENTRY
```

```
SUBROUTINE DELNODE(PNODE)

PURPOSE:   DELETE A NODE FROM THE NODE QUEUE

PARAMETERS:
   PNODE    ==> POINTER TO NODE TO DELETE

SET NODE TYPE TO BLANKS
REMOVE POINTERS TO COMMANDER AND IT'S ALTERNATES
REMOVE ALL ENTRIES FROM NODE'S SUBORDINATE QUEUE AND RELEASE
 MEMORY SPACE FOR FUTURE USE.
REMOVE ALL ENTRIES FROM NODE'S NETWORK QUEUE AND RELEASE
 MEMORY SPACE FOR FUTURE USE.
REMOVE MAIN NODE FROM QUEUE; MARK MAIN NODE AS DELETED;
 RESORT MAIN NODE INTO QUEUE
```

```
SUBROUTINE DISPLAY_LIMITS(MAXLINE,LINE,BOTTOM,BOTLINE,POS)

PURPOSE:  DISPLAY NODE MESSAGE INPUT AND OUTPUT LIMITS IN
SCROLLED AREA. START WITH ENTRY POINTED TO BY PARAMETER POS)

PARAMETERS:
   MAXLINE ==> MAXIMUM NUMBER OF LINES THAT CAN BE DISPLAYED
               IN THE SCROLLED AREA AT ONE TIME.
   LINE    ==> LINE NUMBER OF THE CURRENT LINE OF THE
               SCROLLED AREA.
   BOTTOM  ==> POINTER TO THE LAST NODE THAT IS DISPLAYED
               IN THE SCROLLED AREA.
   BOTLINE ==> LINE NUMBER OF THE LAST NODE THAT IS DISPLAYED
               IN THE SCROLLED AREA.
   POS     ==> POINTER TO NODE THAT IS DISPLAYED ON THE
               CURRENT LINE OF THE SCROLLED AREA.


FMS TERMINATOR CODE
```

```
SUBROUTINE DISPLAY_LINKS(MAXLINE1,LINE1,BOTLINE1,MAXLINE2,

PURPOSE:  DISPLAY COMMUNICATION LINK DICTIONARY AND COMMUNICAT
LINK QUEUE IN SCROLLED AREAS. START WITH ENTRIES POINTED TO BY
POS1 AND POS2, RESPECTIVELY.

PARAMETERS:
   MAXLINE1 ==> MAXIMUM NUMBER OF LINES THAT CAN BE DISPLAYED
                IN THE SCROLLED AREA (FOR LINK DICTIONARY) AT
                ONE TIME.
   LINE1    ==> LINE NUMBER OF THE CURRENT LINE OF THE
                SCROLLED AREA (FOR LINK DICTIONARY).
   BOTLINE1 ==> LINE NUMBER OF THE LAST NODE THAT IS DISPLAYED
                IN THE SCROLLED AREA (FOR LINK DICTIONARY).
   MAXLINE2 ==> MAXIMUM NUMBER OF LINES THAT CAN BE DISPLAYED
                IN THE SCROLLED AREA (FOR LINK QUEUE) AT ONE
                TIME.
   LINE2    ==> LINE NUMBER OF THE CURRENT LINE OF THE
                SCROLLED AREA (FOR LINK QUEUE).
   BOTLINE2 ==> LINE NUMBER OF THE LAST NODE THAT IS DISPLAYED
                IN THE SCROLLED AREA (FOR LINK QUEUE).


FMS TERMINATOR CODE


DISPLAY COMMUNICATIONS LINK DICTIONARY


DISPLAY COMMUNICATIONS LINK QUEUE
```

```
SUBROUTINE DISPLAY_MSG(MAXLINE,LINE,BOTTOM,BOTLINE,POS)

PURPOSE:  DISPLAY EXTERNAL MESSAGE QUEUE IN SCROLLED AREA.

PARAMETERS:
   MAXLINE   ==> MAXIMUM NUMBER OF LINES THAT CAN BE DISPLAYED
                 IN THE SCROLLED AREA AT ONE TIME
   LINE      ==> LINE NUMBER OF THE CURRENT LINE OF THE SCROLLE
   BOTTOM    ==> LINE NUMBER OF THE LAST LINE OF THE SCROLLED A
   BOTLINE   ==> LINE NUMBER OF THE LAST NODE THAT IS DISPLAYED
                 IN THE SCROLLED AREA
   POS       ==> POINTER TO THE CURRENT ENTRY OF THE SCROLLED A

FMS TERMINATOR CODE
```

SUBROUTINE DISPLAY_2900(P2900)

PURPOSE: DISPLAY ADDITIONAL DATA LINE FOR EXTERNAL MESSAGE OF TYPE 2900.

PARAMETERS:
P2900 --> POINTER TO ADDITIONAL DATA LINE

SUBROUTINE DISPLAY_3136(MAXLINE,LINE,BOTTOM,BOTLINE,POS)

PURPOSE: DISPLAY ADDITIONAL DATA LINE QUEUE FOR AN EXTERNAL MESSAGE OF TYPE 3136.

PARAMETERS:

```
   MAXLINE  ==> MAXIMUM NUMBER OF LINES THAT CAN BE DISPLAYED
                IN THE SCROLLED AREA AT ONE TIME
   LINE     ==> LINE NUMBER OF THE CURRENT LINE OF THE SCROLLE
   BOTTOM   ==> LINE NUMBER OF THE LAST LINE OF THE SCROLLED A
   BOTLINE  ==> LINE NUMBER OF THE LAST NODE THAT IS DISPLAYED
                IN THE SCROLLED AREA
   POS      ==> POINTER TO THE CURRENT ENTRY OF THE SCROLLED A
```

FMS TERMINATOR CODE

```
SUBROUTINE DMINIT(MEMORY,MPTR,IGBPTR,MAXDM)
*************************************
*   INITIALIZE DYNAMIC MEMORY
*************************************
INITIALIZE MEMORY POINTER
INITIALIZE GARBAGE POINTER
CLEAR DYNAMIC MEMORY
```

SUBROUTINE EXTMSG

PURPOSE: TO CREATE AND EDIT THE EXTERNAL MESSAGE QUEUE

CONSTANT
  MAXLINE1 - NUMBER OF SCROLLED LINES IN THE GENERIC EXTERNAL MESSAGES AREA

SUBROUTINE FILEOUT

PURPOSE: CREATE OUTPUT FILE FOR FURTHER USE BY PROGRAM C3EVA

OPEN OUTPUT FILE
OUTPUT PREAMBLE DOCUMENTATION
DO FOR ALL ENTRIES IN QUEUE
OUTPUT CURRENT LINE
OUTPUT LINE THAT SIGNALS END OF PREAMBLE DOCUMENTATION
OUTPUT HEADER LINE FOR PRINT FLAGS
OUTPUT PRINT CONTROL FLAGS, OPTIONAL OUTPUT TIMES, AND DEBUG
 OUTPUT FLAG
OUTPUT DEBUG START AND STOP TIMES
WRITE INPUT MODE AND HEADER LINE
DO FOR ALL ENTRIES IN NODE QUEUE
GET NODE'S UNIT NUMBER AND TYPE
GET UNIT NUMBER OF COMMANDER
GET UNIT NUMBER OF 1ST ALTERNATE FOR COMMANDER
GET UNIT NUMBER OF 2ND ALTERNATE FOR COMMANDER
SET COMMANDER AND SUBORDINATE FLAGS, APPROPRIATELY
OUTPUT NODES COMMANDER AND ITS ALTERNATES
SET COMMANDER AND SUBORDINATE FLAGS, APPROPRIATELY
DO FOR ALL ENTRIES IN SUBORDINATE QUEUE
GET UNIT NUMBER OF SUBORDINATE
GET UNIT NUMBER OF 1ST ALTERNATE FOR SUBORDINATE
GET UNIT NUMBER OF 2ND ALTERNATE FOR SUBORDINATE
OUTPUT NODES SUBORDINATE AND IT'S ALTERNATES
SET COMMANDER AND SUBORDINATE FLAGS, APPROPRIATELY
DO FOR ALL ENTRIES IN NETWORK QUEUE
GET UNIT NUMBER OF NETWORK NODE
GET UNIT NUMBER OF 1ST ALTERNATE FOR NETWORK NODE
GET UNIT NUMBER OF 2ND ALTERNATE FOR NETWORK NODE
OUTPUT NODES NETWORK NODE AND IT'S ALTERNATES
CLOSE OUTPUT FILE

```
SUBROUTINE FIND(PIN, N, ID, POUT)
*****************************************************
*   FIND A POINTER IN A QUEUE
*****************************************************
*   INPUT
*       PIN - POINTER TO TOP OF QUEUE TO BE SEARCHED
*       N   - OFFSET FROM PIN TO COMPARE
*       ID  - VALUE TO MATCH WITH N
*****************************************************
*   CREATES
*       POUT - POINTER TO DESIRED ELEMENT
*****************************************************
DO FOR ALL QUEUE ELEMENTS
   COMPARE VALUES
GET NEXT ELEMENT
END OF SEARCH
```

SUBROUTINE FINDDIC

PURPOSE: FIND ALL ENTRIES IN NODE DICTIONARY CORRESPONDING TO INPUTTED TYPE.

GETS A STRING FROM THE USER CORRESPONDING TO THE TYPE NUMBER TO SEARCH FOR
FINDS FIRST ENTRY OF INPUTTED TYPE AND STORES POINTER IN TOP
FIND ALL DICTIONARY ENTRIES OF SAME TYPE
STORE POINTER TO LAST ENTRY OF INPUTTED TYPE IN BOTTOM

```
SUBROUTINE FIND_INDIRECT(PIN, N, ID, POUT)
****************************************************
*   FIND A POINTER IN A QUEUE WHERE THE VALUE TO BE
*     SEARCHED FOR IS INDIRECTLY POINTED TO BY PIN+2
****************************************************
*   INPUT
*      PIN - POINTER TO TOP OF QUEUE TO BE SEARCHED
*      N   - OFFSET FROM MEMORY(PIN+2) TO COMPARE
*      ID  - VALUE TO MATCH WITH N
****************************************************
*   CREATES
*      POUT - POINTER TO DESIRED ELEMENT
****************************************************
DO FOR ALL QUEUE ELEMENTS
   COMPARE VALUES
GET NEXT ELEMENT
END OF SEARCH
```

SUBROUTINE FINDNODE

PURPOSE: SEARCH NODE QUEUE FOR USER INPUTTED NODE.

GET NODE NAME TO SEARCH FOR
SEARCH FOR NODE NAME

IF NODE NAME IS NOT IN NODE QUEU THEN ERROR ELSE SET CURRENT POSITION POINTER TO NODE JUST FOUND

```
SUBROUTINE GETNAME(FLDNAM, FLDIDX, FLDTRM, PTEMP)

PURPOSE:  GETS FROM THE USER A VALID NODE NAME.

PARAMETERS:
   FLDNAM ==> NAME OF FIELD TO INPUT FROM
   FLDIDX ==> INDEX OF FIELD
   FLDTRM ==> VALUE OF FIELD TERMINATOR KEY
   PTEMP  ==> POINTER TO NODE

GET NODE NAME FROM USER. IF NULL ENTRY THEN RETURN NILL POINTE
NODE DOESN'T EXIST, THEREFORE, CREATE NEW NODE
ILLEGAL NAME: TRY AGAIN
LEGAL NAME: SAVE NAME, NUMBER AND TYPE
SNAP INTO QUEUE
OUTPUT NODE'S UNIT NUMBER TO APPROPRIATE ID FIELD
```

```
SUBROUTINE GETNEW(PROOT,LENGTH,FLDNAM,FLDIDX,FLDVAL,FLDTRM)

PURPOSE:  GET UNIQUE ENTRY FROM USER.

PARAMETERS:
   PROOT    ==> ROOT OF QUEUE TO GET UNIQUE ENTRY FOR
   LENGTH   ==> OFFSET FROM PROOT TO COMPARE
   FLDNAM   ==> NAME OF FIELD TO INPUT FROM
   FLDIDX   ==> INDEX OF FIELD TO INPUT FROM
   FLDVAL   ==> VALUE INPUTTED FROM FIELD
   FLDTRM   ==> FMS VALUE OF TERMINATOR KEY

GET ENTRY FROM USER.  IF FIELD INDEX IS 0 THEN DON'T PASS
 TO FMS ROUTINE GET.
SEARCH QUEUE FOR ENTRY
```

```
SUBROUTINE GETSCR(FLDNAM,FVALUE,FLDTRM,PTEMP,PTEMP1,PTEMP2)

PURPOSE:  GETS FROM THE USER A SEQUENCE OF VALID NODE NAMES
  FROM A SCROLLED AREA.

PARAMETERS:
   FLDNAM ==> NAME OF FIELD TO INPUT FROM
   FVALUE ==> VALUES INPUTTED INTO FIELDS
   FLDTRM ==> VALUE OF FIELD TERMINATOR KEY
   PTEMP  ==> POINTER TO NODE
   PTEMP1 ==> POINTER TO 1ST ALTERNATE NODE
   PTEMP2 ==> POINTER TO 2ND ALTERNATE NODE

INITIALIZE POINTERS TO ZERO

IF ALL FIELDS ARE BLANK THEN RETURN NILL POINTER.
IF MAIN FIELD IS BLANK BUT ALTERNATE FIELD IS NON-BLANK
THEN SEND ERROR MESSAGE AND GET INPUT AGAIN.

PROCESS NODE NAME

NODE DOESN'T EXIST, THEREFORE, CREATE NEW NODE
ILLEGAL NAME: TRY AGAIN
LEGAL NAME: SAVE NAME, NUMBER AND TYPE
SNAP INTO QUEUE
SET UNIT NUMBER OF MAIN NODE

IF THERE IS A 1ST ALTERNATE THEN PROCESS 1ST ALTERNATE.
IF NO 1ST ALTERNATE AND NO 2ND ALTERNATE THEN BRANCH TO
  OUTPUT SECTION.
IF NO 1ST ALTERNATE BUT 2ND ALTERNATE THEN ERROR - GET
  INPUT AGAIN.


PROCESS 1ST ALTERNATE

NODE DOESN'T EXIST, THEREFORE, CREATE NEW NODE
ILLEGAL NAME: TRY AGAIN
LEGAL NAME: SAVE NAME, NUMBER AND TYPE
SNAP INTO QUEUE
SET UNIT NUMBER OF 1ST ALTERNATE
IF THERE IS A 2ND ALTERNATE THEN PROCESS 2ND ALTERNATE
  ELSE BRANCH TO OUTPUT SECTION

PROCESS 2ND ALTERNATE

NODE DOESN'T EXIST, THEREFORE, CREATE NEW NODE
ILLEGAL NAME: TRY AGAIN
LEGAL NAME: SAVE NAME, NUMBER AND TYPE
SNAP INTO QUEUE
SET UNIT NUMBER OF 2ND ALTERNATE
OUTPUT LINE TO APPROPRIATE SCROLLED AREA
```

```
SUBROUTINE GETTYP(FLDVAL,ITYPE)

PURPOSE:  SEARCHES NODE NAME FOR ANY WORD THAT OCCURS IN NODE
  DICTIONARY.

PARAMETERS:
   FLDVAL ==> NODE NAME
   ITYPE  ==> UNIT TYPE OF FLDVAL

DO FOR EACH WORD IN NODE NAME
GET NEXT WORD FROM STRING. SEARCH DICTIONARY.
FOUND WORD IN DICTIONARY; SAVE TYPE
DID NOT FIND WORD IN DICTIONARY; SET TYPE TO BLANKS
```

```
SUBROUTINE GETWORD(FVALUE, IWORD)

PURPOSE:  FINDS THE FIRST WORD CONTAINED IN A STRING

PARAMETERS:
   (INPUT)
   FVALUE   ==>  STRING CONTAINING WORDS
   (OUTPUT)
   FVALUE   ==>  INITIAL VALUE WITH 1ST WORD REMOVED
   IWORD    ==>  1ST WORD CONTAINED IN FVALUE

IF ALL BLANKS THEN NO WORD IN STRING
REMOVE LEADING BLANKS
IF LOCATION OF BLANK IS 0 THEN NO BLANK FOUND AND WORD IS ENTI
  STRING.  OTHERWISE, WORD IS ALL LOCATIONS IN STRING PRECEDIN
  LOCATION OF BLANK.
```

```
SUBROUTINE GIMME(NPTR, LEN, ISPACE)

SEGMENT GET VIRTUAL SPACE

PARAMETERS:
   NPTR   ==> POINTER TO BLOCK ALLOCATED
   LEN    ==> LENGTH OF BLOCK TO ALLOCATE
   ISPACE ==> VIRTUAL MEMORY TO GET BLOCK FROM

SEARCH GARBAGE LIST
DO UNTIL LIST ENDS
   IF (SIZE.EQ.LENGTH) THEN
   SET PTR TO FIRST BLOCK
   SNAP GARBAGE PTR
ALLOCATE VIRGIN STORAGE
UPDATE VIR SPACE PTR
STORAGE OVERFLOW
ZERO SPACE BLOCK
END SEGMENT
```

```
SUBROUTINE HCDATA
SUBROUTINE IS NOT IMPLEMENTED YET
```

SUBROUTINE INIT_LIMITS

PURPOSE: TO CREATE A DEFAULT ENTRY IN THE NODE MESSAGE INPUT AND OUTPUT LIMIT QUEUE FOR EACH ENTRY IN THE NODE QUEUE.

CREATE FIRST DEFAULT ENTRY

CREATE ALL OTHER DEFAULT ENTRIES

SUBROUTINE INSTRUC

PURPOSE: PUT FORM CONTAINING INSTRUCTIONS ON SCREEN.

PUT INSTRUCTION FORM ON SCREEN
WAIT FOR USER TO HIT <RETURN>

```
SUBROUTINE INTCHR(VALUE, ISIZE, STRING)

PURPOSE:  CONVERTS AN INTEGER TO ITS ASCII REPRESENTATION.

PARAMETERS:
   VALUE    ==> INTEGER VALUE TO CONVERT
   LENGTH   ==> NUMBER OF DIGITS TO CONVERT
   STRING   ==> ASCII REPRESENTATION OF VALUE
```

SUBROUTINE LIMITS

PURPOSE: USED TO CREATE AND EDIT THE NODE MESSAGE INPUT AND OUTPUT LIMITS DATA SET

FMS TERMINATOR CODES

CONSTANT
MAXLINE1 = NUMBER OF SCROLLED LINES IN THE NODE MESSAGE INP AND OUTPUT LIMITS FIELD

PUT NODE MESSAGE INPUT AND OUTPUT LIMITS FORM IN WORKSPACE

CHECK TO SEE THAT NODE QUEUE EXISTS

IF LIMITS QUEUE IS EMPTY THEN CREATE DEFAULT ENTRIES

PUT COMMUNICATIONS LIMITS FORM ON SCREEN

ALLOW USER TO ADD/CHANGE ENTRIES

GET NAME AND INDEX OF CURRENT FIELD. SAVE OLD VALUE. SAVE OLD NODE NUMBER

GET ENTRY FROM USER

IF ENTRY EQUALS OLD VALUE OR BLANKS THEN NO CHANGE - BRANCH TO PROCESS FIELD TERMINATOR

ADDING NEW ENTRY

VALIDATE THAT INPUTTED NODE EXISTS IN NODE QUEUE

CREATE NEW ENTRY FOR NODE MESSAGE INPUT AND OUTPUT LIMITS Q

CHANGING EXISTING ENTRY

VALIDATE ANY CHANGES TO NODE IDENTIFICATION

SAVE CHANGED VALUES

VALIDATE THAT INPUT HOLD <= INPUT KILL AND OUTPUT HOLD <= OUTPUT KILL

PROCESS FIELD TERMINATOR

IF USER INPUTTED A NEW UNIT NUMBER THEN SKIP OVER UNIT NAME FIELD.

SUBROUTINE MENU

PURPOSE: ALLOWS USER TO SELECT WHICH FUNCTIONS ARE IMPLEMENT IN THE PREPROCESSOR.

PUT MAIN MENU ON SCREEN
GET MODE FROM USER AND CONVERT TO INTEGER
GET OPTION FROM USER AND CONVERT TO INTEGER

SUBROUTINE NODE_C

PURPOSE: USED TO CREATE THE NODE DATA SET.

FMS TERMINATOR CODES

CONSTANTS

MAXLINE1 - NUMBER OF SCROLLED LINES IN THE SUBORDINATES FIE
MAXLINE2 - NUMBER OF SCROLLED LINES IN THE NETWORK NODES FI

IF NODE DICTIONARY IS EMPTY THEN CANNOT CREATE NODES

GET MAIN NODE

NODE NAME MUST BE UNIQUE AND MUST CONTAIN ONE WORD WHICH
IS IN THE NODE DICTIONARY.

STORE NAME, NUMBER, AND TYPE
SNAP INTO QUEUE

PROCESS FIELD TERMINATOR

GET COMMANDER

SAVE POINTER TO COMMANDER

PROCESS FIELD TERMINATOR

GET 1ST ALTERNATE FOR COMMANDER

IF NO COMMANDER THEN SEND MESSAGE; REMOVE 1ST ALTERNATE FROM
SCREEN; BRANCH TO GET COMMANDER
SAVE POINTER TO 1ST ALTERNATE

PROCESS FIELD TERMINATOR

GET 2ND ALTERNATE FOR COMMANDER

IF NO COMMANDER THEN SEND MESSAGE; REMOVE 2ND ALTERNATE FROM SCREEN; BRANCH TO GET COMMANDER
IF NO 1ST ALTERNATE THEN SEND MESSAGE; REMOVE 2ND ALTERNATE FROM SCREEN; BRANCH TO GET 1ST ALTERNATE
SAVE POINTER TO 2ND ALTERNATE

PROCESS FIELD TERMINATOR

GET SUBORDINATE AND ITS ALTERNATES

GET USER ENTRY
VERIFY ENTRIES
IF PSUB IS 0 THEN NULL ENTRY, THEREFORE, NO MORE SUBORDINATES
ADD SUBORDINATE TO BOTTOM OF QUEUE

PROCESS FIELD TERMINATOR

GET NETWORK NODES

GET USER ENTRY
VERIFY ENTRIES
IF PCOM IS 0 THEN NULL ENTRY, THEREFORE, NO MORE NETWORK NODES
ADD NETWORK NODE TO BOTTOM OF QUEUE

PROCESS FIELD TERMINATOR

SUBROUTINE NODE_E

PURPOSE: USED TO EDIT THE NODE DATA SET.

FMS TERMINATOR CODES

CONSTANTS

MAXLINE1 - NUMBER OF SCROLLED LINES IN THE SUBORDINATES FIE
MAXLINE2 - NUMBER OF SCROLLED LINES IN THE NETWORK NODES FI

********** EDIT MODE **********

LOAD NODE FORM INTO WORKSPACE
PRINT MAIN NODE AND ID
PRINT COMMANDER AND ID
PRINT 1ST ALTERNATE FOR COMMANDER AND ID
PRINT 2ND ALTERNATE FOR COMMANDER AND ID
PRINT SUBORDINATES AND THEIR ALTERNATES AND ID'S
PRINT OTHER NETWORK NODES AND THEIR ALTERNATES AND ID'S
PUT NODE FORM ON SCREEN

GET NAME AND INDEX OF CURRENT FIELD. SAVE OLD VALUE.

GET ENTRY FROM USER

IF ENTRY EQUALS OLD VALUE OR BLANKS THEN NO CHANGE - BRANCH TO
FIELD TERMINATOR

CHANGE MAIN NODE NAME

MAIN NODE NAME MUST BE UNIQUE
DETERMINE IF NAME IS VALID, I.E. UNIT WORD OCCURS IN DICTIONAR
LEGAL NAME - SAVE NAME AND TYPE

CHANGE COMMANDER OR IT'S ALTERNATES

LENGTH IS THE OFFSET FROM CURPOS FOR THE POINTER FOR
EITHER THE COMMANDER OR IT'S ALTERNATES

PROCESS BLANK ENTRY

IF ENTRY WAS IN COMMANDER FIELD:
   IF NO ALTERNATES THEN IT IS LEGAL TO DELETE COMMANDER.
   IF ALTERNATES EXIST THEN SEND MESSAGE, PUT OLD VALUE FOR
   COMMANDER BACK ON SCREEN, AND BRANCH TO PROCESS FIELD
   TERMINATOR KEY.
IF ENTRY WAS IN 1ST ALT. FOR COMMANDER FIELD:
   IF NO 2ND ALTERNATE THEN IT IS LEGAL TO DELETE 1ST ALTERNAT
   IF 2ND ALTERNATE EXISTS THEN SEND MESSAGE, PUT OLD VALUE FO
   1ST ALT. BACK ON SCREEN, AND BRANCH TO PROCESS FIELD TERMIN
   KEY.

PROCESS A NON-BLANK ENTRY

IF ENTRY WAS IN 1ST ALT. FOR COMMANDER FIELD:
   IF COMMANDER EXISTS THEN IT IS LEGAL TO ADD 1ST ALTERNATE.
   IF NO COMMANDER THEN SEND MESSAGE, REMOVE 1ST ALTERNATE FRO
   SCREEN, AND BRANCH TO PROCESS FIELD TERMINATOR KEY.
IF ENTRY WAS IN 2ND ALT. FOR COMMANDER FIELD:
   IF 1ST ALTERNATE EXISTS THEN IT IS LEGAL TO ADD 2ND ALTERNA
   IF NO 1ST ALTERNATE THEN SEND MESSAGE, REMOVE 2ND ALTERNATE
   FROM SCREEN, AND BRANCH TO PROCESS FIELD TERMINATOR KEY.
DETERMINE IF NAME IS UNIQUE
ENTRY DOESN'T EXIST; CHECK IF VALID NAME, I.E. UNIT WORD
 OCCURS IN DICTIONARY
CREATE NODE
SAVE POINTER TO NODE
WRITE NEW ID TO SCREEN AND BRANCH TO PROCESS FIELD TERMINATOR

CHANGE SUBORDINATE AND ALTERNATES

GET SUBORDINATE NAME FROM FIELD
GET 1ST ALTERNATE FROM FIELD
GET 2ND ALTERNATE FROM FIELD
VERIFY ENTRIES

ADDING NEW ENTRIES

CREATE NEW ENTRY FOR SUBORDINATE QUEUE
SET BACK POINTER OF NEW ENTRY TO LAST ENTRY OF QUEUE
SET SUBORDINATE AND ALTERNATE POINTERS TO APPROPRIATE VALUES
IF QUEUE IS EMPTY THEN ADD NEW ENTRY TO TOP ELSE ADD TO BOTTOM
BRANCH TO PROCESS FIELD TERMINATOR KEY

CHANGING EXISTING ENTRIES

BRANCH TO PROCESS FIELD TERMINATOR KEY

CHANGE NETWORK NODES AND ALTERNATES

GET NETWORK NAME FROM FIELD
GET 1ST ALTERNATE FROM FIELD
GET 2ND ALTERNATE FROM FIELD
VERIFY ENTRIES

ADDING NEW ENTRIES

CREATE NEW ENTRY FOR NETWORK QUEUE
SET BACK POINTER OF NEW ENTRY TO LAST ENTRY OF QUEUE
SET NETWORK AND ALTERNATE POINTERS TO APPROPRIATE VALUES
IF QUEUE IS EMPTY THEN ADD NEW ENTRY TO TOP ELSE ADD TO BOTTOM
BRANCH TO PROCESS FIELD TERMINATOR KEY

CHANGING EXISTING ENTRIES

PROCESS FIELD TERMINATOR

SUBROUTINE NODEDIC_C

PURPOSE: CREATE ENTRIES WITHIN THE NODE DICTIONARY.

FMS TERMINATOR CODES

GET TYPE
GET NAMES
SORT BY TYPE
SORT BY NAME

PROCESS FIELD TERMINATOR

SUBROUTINE NODEDIC_E

PURPOSE: EDIT ENTRIES WITHIN THE NODE DICTIONARY.

FMS TERMINATOR CODES

```
SET TOP TO FIRST ENTRY IN DICTIONARY. SET BOTTOM TO LAST ENTRY
  THAT HAS SAME TYPE AS FIRST ENTRY.
INITIALIZE FIELDS
PUT NODE DICTIONARY FORM ON SCREEN
      GET POINTER TO CURRENT NAME
         REMOVE EXISTING NAME
         CHANGE EXISTING NAME
   ADD NEW NAME
SORT BY TYPE
SORT BY NAME
IF NEW NAME WAS ADDED BEFORE FIRST ENTRY OF IT'S TYPE
 THEN UPDATE POINTER TO TOP
IF NEW NAME WAS ADDED AFTER LAST ENTRY OF IT'S TYPE
 THEN UPDATE POINTER TO BOTTOM
CHANGE EXISTING TYPE
RESORT TYPE
PROCESS FIELD TERMINATOR
```

```
SUBROUTINE POUT(MEMORY, NUM, LENGTH)

PURPOSE:
   PRINT OUT CONTENTS OF VIRTUAL MEMORY

PARAMETERS:
   MEMORY   ==> VIRTUAL MEMORY TO PRINT
   NUM      ==> NUMBER OF LINES TO PRINT
   LENGTH   ==> NUMBER OF VALUES TO PRINT PER LINE
```

SUBROUTINE PREAM_C

PURPOSE: ALLOW USER TO CREATE THE PREAMBLE DOCUMENTATION.

FMS TERMINATOR KEYS

CLEAR DISPLAY AND PUT THE FORM FOR THE PREAMBLE ON THE SCREEN
INITIALIZE THE CURRENT FIELD NAME AND INDEX
DO UNTIL USER HITS <RETURN> KEY
GET USER ENTRY FROM CURRENT FIELD
FIRST ENTRY - SAVE VALUE; SET ROOT POINTER
QUEUE NOT EMPTY - SAVE VALUE; ADD TO BOTTOM OF QUEUE

PROCESS FIELD TERMINATOR

IF <TAB> KEY THEN SET CURRENT FIELD TO NEXT FIELD

SUBROUTINE PREAM_E

PURPOSE: ALLOW USER TO EDIT THE PREAMBLE DOCUMENTATION.

FMS TERMINATOR KEYS

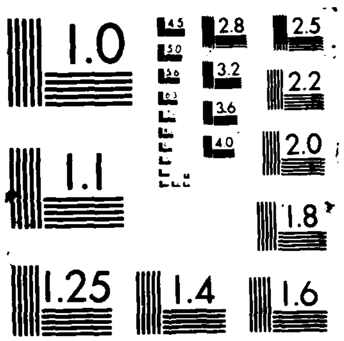
IF QUEUE IS EMPTY THEN CANNOT EDIT; BRANCH TO END OF SUBROUTINE
LOAD PREAMBLE FORM INTO WORKSPACE
INITIALIZE POINTER TO TOP OF QUEUE; INITIALIZE CURRENT FIELD NAME AND INDEX
PUT EXISTING DOCUMENTATION INTO WORKSPACE
PUT THE FORM FOR THE PREAMBLE ON THE SCREEN
SET POINTER TO TOP OF QUEUE AND FIELD INDEX TO ONE
DO UNTIL USER HITS <RETURN> KEY
GET USER ENTRY FROM CURRENT FIELD
NEW LINE - SAVE VALUE; ADD ENTRY TO BOTTOM OF QUEUE; UPDATE POINTER TO LAST LINE; BRANCH TO PROCESS FIELD TERMINATOR
OLD LINE - UPDATE VALUE

PROCESS FIELD TERMINATOR

IF <TAB> KEY THEN SET CURRENT FIELD TO NEXT FIELD
IF <BACKSPACE> KEY THEN SET CURRENT FIELD TO PREVIOUS FIELD

AD-A186 370   C3EVAL MODEL DEVELOPMENT--1986 VOLUME 2 PROGRAMMERS' MANUAL(U) INSTITUTE FOR DEFENSE ANALYSES ALEXANDRIA VA R F ROBINSON ET AL. APR 87 IDA-P-1978-VOL-2   2/4

UNCLASSIFIED   IDA/HQ-86-31573 MDA903-84-C-0031   F/G 12/5   NL


1.0
4.5
5.0
5.6
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

SUBROUTINE QPRINT(PROMPT)

PURPOSE: PRINT OUT CONTENTS OF NODE DICTIONARY QUEUE

PARAMETERS:
PROMPT ==> HEADER STRING TO OUTPUT ALONG WITH CONTENTS OF NODE DICTIONARY QUEUE.

OUTPUT HEADER PROMPT
DO FOR ALL ENTRIES IN NODE DICTIONARY QUEUE
OUTPUT FORWARD AND BACKWARD POINTERS FOR BOTH TYPE AND NAME SORTING. OUTPUT TYPE AND NAME.

SUBROUTINE RANGE(MIN, MAX, STRING, LENGTH, LEGAL)

PURPOSE: TO DETERIMINE IF VALUE IS BETWEEN MINIMUM AND MAXIMUM VALUES.

```
SUBROUTINE RELEAS(NPTR, LEN, ISPACE)

PURPOSE:  SEGMENT RELEASE PUTS STORAGE ON GARBAGE LIST

PARAMETERS:
   NPTR   ==> POINTER TO BLOCK TO RELEASE
   LEN    ==> LENGTH OF BLOCK TO RELEASE
   ISPACE ==> VIRTUAL MEMORY BLOCK IS CONTAINED IN

CHECK BAD PTR, LEN
DO UNTIL NO GARBAGE EQUAL LENGTH
END DO
SNAP IN SPACE
GARBAGE LENGTH NOT KNOWN
PUT STORAGE ON GARBAGE LIST
END SEGMENT
```

```
SUBROUTINE RESTORE
**********************************************************
*   DYNAMIC MEMORY AND COMMON VALUES ARE READ FROM A FILE
*   INTO MEMORY AS PHYSICAL STRUCTURES. THIS FILE IS
*   CREATED BY SUBROUTINE STORE.
**********************************************************
RESTORE COMMON VARIABLES
RESTORE DYNAMIC MEMORY
```

```
SUBROUTINE RETSC(START, FINISH, VALUE)

PURPOSE:  RETURNS THE VALUES THAT ARE CURRENTLY STORED IN
  CONSECUTIVE FIELDS.

PARAMETERS:
  START    => FIELD NUMBER OF FIRST FIELD TO READ
  FINISH   => FIELD NUMBER OF LAST FIELD TO READ
  (OUTPUT)
  VALUE    => CONCATENATED STRING OF ALL RETURNED FIELD STRINGS
```

SUBROUTINE RULES
SUBROUTINE IS NOT IMPLEMENTED YET

```
SUBROUTINE SAVE
**********************************************************
*   DYANMIC MEMORY AND COMMON VALUES ARE WRITTEN TO A FILE
*   AS PHYSICAL STRUCTURES. THIS FILE MAY BE USED TO
*   RESTART THE SIMULATION AT THE POINT WHERE IT LEFT OFF.
**********************************************************
SAVE COMMON VARIABLES
SAVE DYNAMIC MEMORY
```

```
SUBROUTINE SCRBK(FLDNAM,POS,BOTTOM,LINE,BOTLINE,MAXLINE,FVALUE

PURPOSE:   SCROLL A SCROLLED AREA BACKWARD

PARAMETERS:
   FLDNAM  ==> NAME OF FIELD IN SCROLLED AREA TO SCROLL
   POS     ==> POINTER TO NODE THAT IS DISPLAYED ON THE
               CURRENT LINE OF THE SCROLLED AREA.
   BOTTOM  ==> POINTER TO THE LAST NODE THAT IS DISPLAYED
               IN THE SCROLLED AREA.
   LINE    ==> LINE NUMBER OF THE CURRENT LINE OF THE
               SCROLLED AREA.
   BOTLINE ==> LINE NUMBER OF THE LAST NODE THAT IS DISPLAYED
               IN THE SCROLLED AREA.
   MAXLINE ==> MAXIMUM NUMBER OF LINES THAT CAN BE DISPLAYED
               IN THE SCROLLED AREA AT ONE TIME.
   FVALUE  ==> LINE TO OUTPUT TO SCROLLED AREA IF CURRENT LINE
               IS THE TOP LINE OF THE SCROLLED AREA

IF NO LAST LINE OF SCROLLED AREA THEN NO NODES ARE DISPLAYED
IF CURRENT LINE IS BLANK THEN CURRENT LINE IS BELOW LAST
  DISPLAYED LINE.  THEREFORE, SET CURRENT LINE TO LAST
  DISPLAYED LINE.
IF CURRENT NODE IS TOP OF QUEUE THEN SEND MESSAGE ELSE
 SET CURRENT NODE TO PREVIOUS NODE

*** TOP OF SCROLLED AREA ***

WRITE SCROLLED LINE TO SCREEN

*** NOT TOP OF SCROLLED AREA ***

MOVE CURRENT LINE OF SCROLLED AREA UP ONE LINE
```

```
SUBROUTINE SCRFWD(FLDNAM,POS,BOTTOM,LINE,BOTLINE,MAXLINE,FVALU

PURPOSE:    SCROLL A SCROLLED AREA FORWARD

PARAMETERS:
   FLDNAM  --> NAME OF FIELD IN SCROLLED AREA TO SCROLL
   POS     --> POINTER TO NODE THAT IS DISPLAYED ON THE
                CURRENT LINE OF THE SCROLLED AREA.
   BOTTOM  --> POINTER TO THE LAST NODE THAT IS DISPLAYED
               IN THE SCROLLED AREA.
   LINE    --> LINE NUMBER OF THE CURRENT LINE OF THE
               SCROLLED AREA
   BOTLINE --> LINE NUMBER OF THE LAST NODE THAT IS DISPLAYED
               IN THE SCROLLED AREA.
   MAXLINE --> MAXIMUM NUMBER OF LINES THAT CAN BE DISPLAYED
               IN THE SCROLLED AREA AT ONE TIME.
   FVALUE  --> LINE TO OUTPUT TO SCROLLED AREA IF CURRENT LINE
               IS THE LAST LINE OF THE SCROLLED AREA

IF NO LAST LINE OF SCROLLED AREA THEN NO NODES ARE DISPLAYED
IF CURRENT LINE IS BLANK THEN DO NOT ALLOW TO SCROLL FORWARD
 I.E. ONLY ONE BLANK LINE AT BOTTOM OF SCROLLED AREA CAN BE
 USED TO INPUT NEW ENTRIES.

*** BOTTOM OF SCROLLED AREA ***

   IF ADDING BLANK LINE FOR NEW ENTRY THEN
   UPDATE THE LINE NUMBER FOR THE BOTTOM ENTRY
   WRITE SCROLLED LINE TO SCREEN

*** NOT BOTTOM OF SCROLLED AREA

MOVE CURRENT LINE OF SCROLLED AREA DOWN ONE LINE
```

SUBROUTINE SCRLINE(POS,FVALUE)

PURPOSE: CREATE STRING TO OUTPUT TO THE CURRENT LINE OF THE SCROLLED AREA FOR NODE QUEUE

PARAMETERS:

POS ==> POINTER TO NODE TO CONVERT TO GET FIELD VALUES FOR SCROLLED LINE.

FVALUE ==> FIELD VALUES FOR SCROLLED LINE.

NULL ENTRY, THEREFORE INITIALIZE TO BLANKS

INITIALIZE PORTION OF SCROLLED LINE THAT CONTAINS MAIN NODE AND IT'S ID

INITIALIZE PORTION OF SCROLLED LINE THAT CONTAINS 1ST ALTERNATE AND IT'S ID

INITIALIZE PORTION OF SCROLLED LINE THAT CONTAINS 2ND ALTERNATE AND IT'S ID

```
SUBROUTINE SCRLINE2(POS,FVALUE)

PURPOSE:   CREATE STRING TO OUTPUT TO THE CURRENT LINE OF
   THE SCROLLED AREA FOR NODE MESSAGE INPUT AND OUTPUT LIMITS

PARAMETERS:
   POS      ==> POINTER TO NODE TO CONVERT TO GET FIELD
                VALUES FOR SCROLLED LINE.

   FVALUE   ==> FIELD VALUES FOR SCROLLED LINE.


NULL ENTRY, THEREFORE INITIALIZE TO DEFAULT VALUES
```

SUBROUTINE SCRLINE3(POS, FVALUE)

PURPOSE: CREATE STRING TO OUTPUT TO THE CURRENT LINE OF THE SCROLLED AREA FOR COMMUNICATIONS LINK DICTIONARY

PARAMETERS:

POS ==> POINTER TO NODE TO CONVERT TO GET FIELD VALUES FOR SCROLLED LINE.

FVALUE ==> FIELD VALUES FOR SCROLLED LINE.

NULL ENTRY, THEREFORE INTIALIZE TO DEFAULTS

```
SUBROUTINE SCRLINE4(POS, FVALUE)

PURPOSE:   CREATE STRING TO OUTPUT TO THE CURRENT LINE OF
   THE SCROLLED AREA FOR COMMUNICATIONS LINK QUEUE

PARAMETERS:
   POS      ==>  POINTER TO NODE TO CONVERT TO GET FIELD
                 VALUES FOR SCROLLED LINE.

   FVALUE   ==>  FIELD VALUES FOR SCROLLED LINE.


   NULL ENTRY, THEREFORE INTIALIZE TO DEFAULT VALUES
```

```
SUBROUTINE SCRLINE5(POS, FVALUE)

PURPOSE:   CREATE STRING TO OUTPUT TO THE CURRENT LINE OF
   THE SCROLLED AREA FOR EXTERNAL MESSAGES.

PARAMETERS:
   POS      ==>  POINTER TO NODE TO CONVERT TO GET FIELD
                 VALUES FOR SCROLLED LINE.

   FVALUE   ==>  FIELD VALUES FOR SCROLLED LINE.


NULL ENTRY, THEREFORE INITIALIZE TO DEFAULT VALUES
```

SUBROUTINE SCRLINE6(POS, FVALUE)

PURPOSE: CREATE STRING TO OUTPUT TO THE CURRENT LINE OF THE SCROLLED AREA FOR ADDITIONAL DATA FOR AN EXTERNAL MESSAGE OF TYPE 2900

PARAMETERS:

POS ==> POINTER TO NODE TO CONVERT TO GET FIELD VALUES FOR SCROLLED LINE.

FVALUE ==> FIELD VALUES FOR SCROLLED LINE.

NULL ENTRY, THEREFORE INITIALIZE TO DEFAULT VALUES

SUBROUTINE SCRLINE7(POS, FVALUE)

PURPOSE: CREATE STRING TO OUTPUT TO THE CURRENT LINE OF THE SCROLLED AREA FOR ADDITIONAL DATA FOR AN EXTERNAL MESSAGE OF TYPE 3136

PARAMETERS:

POS ==> POINTER TO NODE TO CONVERT TO GET FIELD VALUES FOR SCROLLED LINE.

FVALUE ==> FIELD VALUES FOR SCROLLED LINE.

NULL ENTRY, THEREFORE INITIALIZE TO DEFAULT VALUES

```
SUBROUTINE SEARCH_LIMITS(LINE1,BOTLINE1,MAXLINE1)

PURPOSE:  SEARCH FOR AN ENTRY IN THE NODE MESSAGE INPUT AND
  OUTPUT LIMITS QUEUE. THERE ARE 5 WAYS TO SEARCH.

PARAMETERS:
   LINE1     ==> LINE NUMBER OF THE CURRENT LINE OF THE
                 SCROLLED AREA
   BOTLINE1 ==> LINE NUMBER OF THE LAST NODE THAT IS DISPLAYED
                 IN THE SCROLLED AREA.
   MAXLINE1 ==> MAXIMUM NUMBER OF LINES THAT CAN BE DISPLAYED
                 IN THE SCROLLED AREA AT ONE TIME.
```

SUBROUTINE SETFLAG

PURPOSE: CREATE FLAG NODE. INITIALIZE ALL VALUES TO ZERO.

GET VIRTUAL MEMORY SPACE
INITIALIZE 20 PRINT FLAGS
INITIALIZE THE 3 OPTIONAL PRINT MODIFIERS
INITIALIZE DEBUG PRINT FLAG, DEBUG TIME ON, AND DEBUG TIME OFF

SUBROUTINE SIMCTRL

PURPOSE: ALLOW USER TO CHANGE VALUES OF PRINT FLAGS.

FMS TERMINATOR KEYS

LOAD SIMULATION CONTROL FORM INTO WORKSPACE

PUT VALUES IN APPROPRIATE FIELDS

PUT SIMULATION CONTROL FORM ON SCREEN

GET USER ENTRY

ENTRY WAS A PRINT CONTROL FLAG

IF ENTRY NOT EITHER 'O' OR '1' THEN ILLEGAL; RESTORE OLD VALUE
 GET ENTRY AGAIN
LEGAL ENTRY - SAVE NEW VALUE; BRANCH TO PROCESS FIELD TERMINAT

ENTRY WAS AN OPTIONAL OUTPUT TIME - SAVE NEW TIME; BRANCH TO
 PROCESS FIELD TERMINATOR

ENTRY WAS DEBUG OUTPUT FLAG

IF ENTRY NOT EITHER 'O' OR '1' THEN ILLEGAL; RESTORE OLD VALUE
 GET ENTRY AGAIN
LEGAL ENTRY - SAVE NEW VALUE; BRANCH TO PROCESS FIELD TERMINAT

ENTRY WAS DEBUG ON/OFF TIME - SAVE TIME; BRANCH TO PROCESS FIE
 TERMINATOR

PROCESS FIELD TERMINATOR

SUBROUTINE SNAPQ(PTR, NWORD, PINS, PINE)

PURPOSE:
INSERT AN ENTIRE QUEUE OF RECORDS INTO ANOTHER QUEUE OF RECORDS. ASSUMES THAT THERE IS A CORRESPONDING BACK POINTER FOR THE FORWARD POINTER. ASSUMES BACK POINTER IS OFFSET FROM ITS FORWARD POINTER BY 1.

PARAMETERS:

```
PTR    ==> ROOT OF BASE QUEUE
NWORD  ==> OFFSET FROM PTR TO SORT ON
PINS   ==> POINTER TO TOP OF QUEUE BEING ADDED
PINE   ==> POINTER TO BOTTOM OF QUEUE BEING ADDED
```

EMPTY QUEUE - MAKE FIRST RECORD

INSERT BEFORE RECORD

INSERT BEFORE 1ST RECORD

INSERT AFTER LAST RECORD

```
SUBROUTINE UNSNAP(NROOT, NPTR, ISPACE)

PURPOSE:  REMOVES AN ENTRY FROM QUEUE

PARAMETERS:
   NROOT  --> POINTER TO ROOT OF QUEUE
   NPTR   --> POINTER TO NODE TO REMOVE FROM QUEUE
   ISPACE --> VIRTUAL MEMORY SPACE QUEUE IS STORED IN

SET FORWARD AND BACK POINTERS
IF NODE BEING REMOVED IS NOT 1ST NODE IN QUEUE THEN SET
 FORWARD POINTER OF PREVIOUS NODE TO NEXT NODE.
IF NODE BEING REMOVED IS 1ST NODE IN QUEUE THEN SET ROOT
 TO NEXT NODE
IF NODE BEING REMOVED IS NOT LAST NODE IN QUEUE THEN SET
 BACK POINTER OF NEXT NODE TO PREVIOUS NODE.
```

```
SUBROUTINE UPD_LIMITS(PLIMITS)

PURPOSE:  RETRIEVE VALUES FROM THE CURRENT LINE OF THE
  NODE MESSAGE INPUT AND OUTPUT LIMITS SCROLLED AREA.
  UPDATE CURRENT ENTRY IN QUEUE TO REFLECT THESE VALUES.
  NOTE: VALUES FOR NODE'S IDENTIFICATION ARE NOT UPDATED
  IN THIS SUBROUTINE.

PARAMETERS:
  PLIMITS  =>  POINTER TO THE NODE THAT IS DISPLAYED ON
               THE CURRENT LINE OF THE SCROLLED AREA.

GET NEW VALUE FOR TIME
GET NEW VALUE FOR NODE MESSAGE INPUT LIMIT - HOLD
GET NEW VALUE FOR NODE MESSAGE INPUT LIMIT - KILL
GET NEW VALUE FOR NODE MESSAGE OUTPUT LIMIT - HOLD
GET NEW VALUE FOR NODE MESSAGE OUTPUT LIMIT - KILL
GET NEW VALUE FOR RANDOM DISTRIBUTION - CAPACITY
GET NEW VALUE FOR RANDOM DISTRIBUTION - DELETE
GET NEW VALUE FOR RANDOM DISTRIBUTION - CAS
```

```
SUBROUTINE UPD_MSG(FLDNAM,FLDIDX,FLDVAL,FVALO,FLDTRM,NEWFORM,

PURPOSE:  USER'S CURRENT FIELD WAS WITHIN AREA FOR EXTERNAL ME
  PROCESS ANY CHANGES USER MADE TO FIELD AND PROCESS FIELD TER
  IF APPROPRIATE.

PARAMETERS:
  FLDNAM    => NAME OF THE CURRENT FIELD
  FLDIDX    => INDEX OF THE CURRENT FIELD
  FLDVAL    => VALUE AT THE CURRENT FIELD
  FVALO     => OLD VALUE AT THE CURRENT FIELD
  FLDTRM    => VALUE OF FIELD TERMINATOR KEY
  NEWFORM   => FLAG FOR DISPLAYING ADDITIONAL DATA FORM
               0 -> NO NEW FORM        1 -> UPDATE NEW FORM AREA
  DONE      => FLAG FOR ENDING INPUT FOR EXTERNAL MESSAGES
               0 -> CONTINUE INPUT FOR EXTERNAL MESSAGES
               1 -> RETURN TO MAIN MENU
  MAXLINE1 => MAXIMUM NUMBER OF LINES IN THE SCROLLED AREA FOR
               EXTERNAL MESSAGES
  LINE1     => LINE NUMBER OF THE CURRENT LINE IN THE SCROLLED
               FOR EXTERNAL MESSAGES
  BOTLINE1 => LINE NUMBER OF THE BOTTOM LINE IN THE SCROLLED A
               FOR EXTERNAL MESSAGES


FMS TERMINATOR CODES


FIELD 'SENDTIME'

   CREATING NEW ENTRY
   CHANGING EXISTING SEND TIME

FIELD 'MSGTYPE'

   CREATING NEW ENTRY
   CHANGING EXISTING MESSAGE TYPE

FIELD 'ORIGTYPE'

   CREATING NEW ENTRY
   CHANGING EXISTING ORIGINATING UNIT TYPE

FIELD 'DESTID'

   CREATING NEW ENTRY
   VALIDATE THAT UNIT ID EXISTS IN NODE QUEUE
   CHANGING EXISTING DESTINATION UNIT ID
   VALIDATE THAT UNIT ID EXISTS IN NODE QUEUE
```

```
FIELD 'CREATETIME'

   CREATING NEW ENTRY
   VALIDATE THAT CREATE TIME <= SEND TIME
   CHANGING EXISTING CREATE TIME
   VALIDATE THAT CREATE TIME <= SEND TIME AND
   THAT CREATE TIME >= REPORT TIME

FIELD 'PRINTFLAG'

   CREATING NEW ENTRY
   CHANGING EXISTING PRINT FLAG

FIELD 'PRIORITY'

   CREATING NEW ENTRY
   CHANGING EXISTING PRIORITY

FIELD 'MAXTIME'

   CREATING NEW ENTRY
   CHANGING EXISTING MAXIMUM NETWORK TIME

PROCESS FIELD TERMINATOR

   SEARCH FOR ENTRY IN MSG QUEUE OR NODE QUEUE
   DELETE ENTRY FROM MSG QUEUE
```

```
SUBROUTINE UPD_2900(FLDNAM,FLDIDX,FLDVAL,FVALO,FLDTRM,DONE)

PURPOSE:  USER'S CURRENT FIELD WAS WITHIN AREA FOR ADDITIONAL
  LINE FOR AN EXTERNAL MESSAGE OF TYPE 2900.  PROCESS ANY CHAN
  USER MADE TO FIELD AND PROCESS FIELD TERMINATOR IF APPROPRIA

PARAMETERS:
  FLDNAM    => NAME OF THE CURRENT FIELD
  FLDIDX    => INDEX OF THE CURRENT FIELD
  FLDVAL    => VALUE AT THE CURRENT FIELD
  FVALO     => OLD VALUE AT THE CURRENT FIELD
  FLDTRM    => VALUE OF FIELD TERMINATOR KEY
  DONE      => FLAG FOR ENDING INPUT FOR EXTERNAL MESSAGES
               0 -> CONTINUE INPUT FOR EXTERNAL MESSAGES
               1 -> RETURN TO MAIN MENU


FMS TERMINATOR CODES


FIELD 'F2900ID'
VALIDATE THAT SUPPORT UNIT ID EXISTS IN NODE QUEUE


FIELD 'F2900EARLY'
VALIDATE THAT EARLIEST SUPPORT TIME <= LATEST SUPPORT TIME


FIELD 'F2900LATE'
VALIDATE THAT LATEST SUPPORT TIME >= EARLIEST SUPPORT TIME


FIELD 'F2900TYPE'


FIELD 'F2900NUMBER'


PROCESS FIELD TERMINATOR

   SEARCH FOR ENTRY
   DELETE ENTRY (SET ALL VALUES TO DEFAULT'S)
```

```
SUBROUTINE UPD_3136(FLDNAM,FLDIDX,FLDVAL,FVALO,FLDTRM,DONE,

PURPOSE:  USER'S CURRENT FIELD WAS WITHIN AREA FOR ADDITIONAL
  LINE FOR AN EXTERNAL MESSAGE OF TYPE 3136.  PROCESS ANY CHAN
  USER MADE TO FIELD AND PROCESS FIELD TERMINATOR IF APPROPRIA

PARAMETERS:
  FLDNAM    => NAME OF THE CURRENT FIELD
  FLDIDX    => INDEX OF THE CURRENT FIELD
  FLDVAL    => VALUE AT THE CURRENT FIELD
  FVALO     => OLD VALUE AT THE CURRENT FIELD
  FLDTRM    => VALUE OF FIELD TERMINATOR KEY
  DONE      => FLAG FOR ENDING INPUT FOR EXTERNAL MESSAGES
               0 -> CONTINUE INPUT FOR EXTERNAL MESSAGES
               1 -> RETURN TO MAIN MENU
  MAXLINE2  => MAXIMUM NUMBER OF LINES IN THE SCROLLED AREA FOR
               ADDITONAL DATA FOR AN EXTERNAL MESSAGE OF TYPE 3
  LINE2     => LINE NUMBER OF THE CURRENT LINE IN THE SCROLLED
               FOR ADDITONAL DATA FOR AN EXTERNAL MESSAGE OF TY
  BOTLINE2  => LINE NUMBER OF THE BOTTOM LINE IN THE SCROLLED A
               FOR ADDITIONAL DATA FOR AN EXTERNAL MESSAGE OF T


FMS TERMINATOR CODES


FIELD 'F3136UNITID'

   CREATING NEW ENTRY
   VALIDATE THAT UNIT ID EXISTS IN NODE QUEUE
   CHANGING EXISTING UNIT ID
   VALIDATE THAT UNIT ID EXISTS IN NODE QUEUE

FIELD 'F3136TIME'

   CREATING NEW ENTRY
   VALIDATE THAT REPORT TIME <= CREATE TIME
   CHANGING EXISTING REPORT TIME
   VALIDATE THAT REPORT TIME <= CREATE TIME

FIELD 'F3136TYPE'

   CREATING NEW ENTRY
   CHANGING EXISTING FOE TYPE
```

FIELD 'F3136POSTURE'

CREATING NEW ENTRY
CHANGING EXISTING FOE POSTURE

FIELD 'F3136FOEID'

CREATING NEW ENTRY
CHANGING EXISTING FOE ID

PROCESS FIELD TERMINATOR

SEARCH FOR ENTRY
DELETE ENTRY

```
SUBROUTINE VALID_NODE(OLDNAME,OLDNUMBER,FLDIDX,PNODE)

PURPOSE:  DETERMINE IF UNIT IDENTIFER IS VALID, I.E. IF
  GAVE NAME ONLY THEN NAME EXISTS IN NODE QUEUE. IF GAVE
  NUMBER ONLY THEN NUMBER EXISTS IN NODE QUEUE. IF GAVE
  NAME AND NUMBER THEN NODE EXISTS IN NODE QUEUE AND NAME
  AND NUMBER MATCH. RETURNS THE POINTER TO THE NODE IF IT
  IS VALID, OTHERWISE, RETURNS A ZERO.

PARAMETERS:
  OLDNAME   ==> OLD FIELD VALUE FOR FIELD 'NAME'
  OLDNUMBER ==> OLD FIELD VALUE FOR FIELD 'NUMBER'
  FLDIDX    ==> FIELD INDEX TO GET VALUES FROM
  (OUTPUT)
  PNODE     ==> POINTER TO THE NODE SPECIFIED BY THE USER


      NEW UNIT NAME AND NUMBER


         ILLEGAL NODE NUMBER


         ILLEGAL, NODE NAME AND NUMBER DON'T MATCH


   NEW NODE NAME ONLY


      NEW NODE NUMBER ONLY


      NO NODE NAME OR NUMBER GIVEN - ERROR


DISPLAY UNIT'S IDENTIFICATION


********  UAR ROUTINES   **********
```

INTEGER FUNCTION VALID1

FIELD COMPLETION UAR CHECKS IF VALUE IS BETWEEN 1
AND THE MAXIMUM FOR THE APPROPRIATE FIELD.

GET FIELD NAME OF CURRENT FIELD
GET VALUE AT CURRENT FIELD
CONVERT VALUE TO INTEGER
GET MAXIMUM LEGAL VALUE FOR CURRENT FIELD
CONVERT MAXIMUM LEGAL VALUE TO INTEGER
IF CURRENT VALUE IS BETWEEN 1 AND MAXIMUM LEGAL VALUE
 THEN RETURN SUCCESS CODE ELSE SEND ERROR MESSAGE AND
 RETURN FAILURE CODE.

III. PROGRAM C3EVAL

There are three distinct elements simulated by C3EVAL. The first is the C3 environment. It consists of a set of nodes (command posts), paths (lines of communication) and processing of messages and combat data. Figure III-1 gives an example of a command network that could be represented in the model. In this example Div units would be designated as the combat level units. Each one has specific ground-based weapon systems (tanks, anti-aircraft artillery, infantry, etc.) specified for its own forces (Blue) and for its opposing force (Red). In addition, a notional air base has aircraft (Blue only) which are requested by air tasking order (ATO) messages for specific combat units at a specified game time interval. These weapon systems and aircraft sorties are the second element.

The third element of the C3EVAL is the combat process. This is modeled by using IDA's matrix method of calculating combat allocation and effectiveness potential. Attrition is calculated by multiplying the effectiveness matrix times each unit's weapon system vector. The sequence of events in the model is:

- Limit input messages,
- Process messages received,
- Create messages based on command post actions,
- Limit output messages,
- Allocate messages to network,
- Put messages into communication status,
- Process requests for CAS,
- Update combat unit's weapon systems, and
- Calculate combat drawdown.

The C3 portion of C3EVAL has been implemented in a user-controlled dynamic memory (DM) environment. This DM is a large single-dimension common area that is segmented, allocated and reused by two simple C3EVAL utilities. The effect of this implementation decision is a code that provides for a vast variation in the number of nodes, paths or messages that may be used from different data bases and scenarios without major parameter changes in the model. (Only the maximum size of DM is modified to provide efficient computer memory usage.)

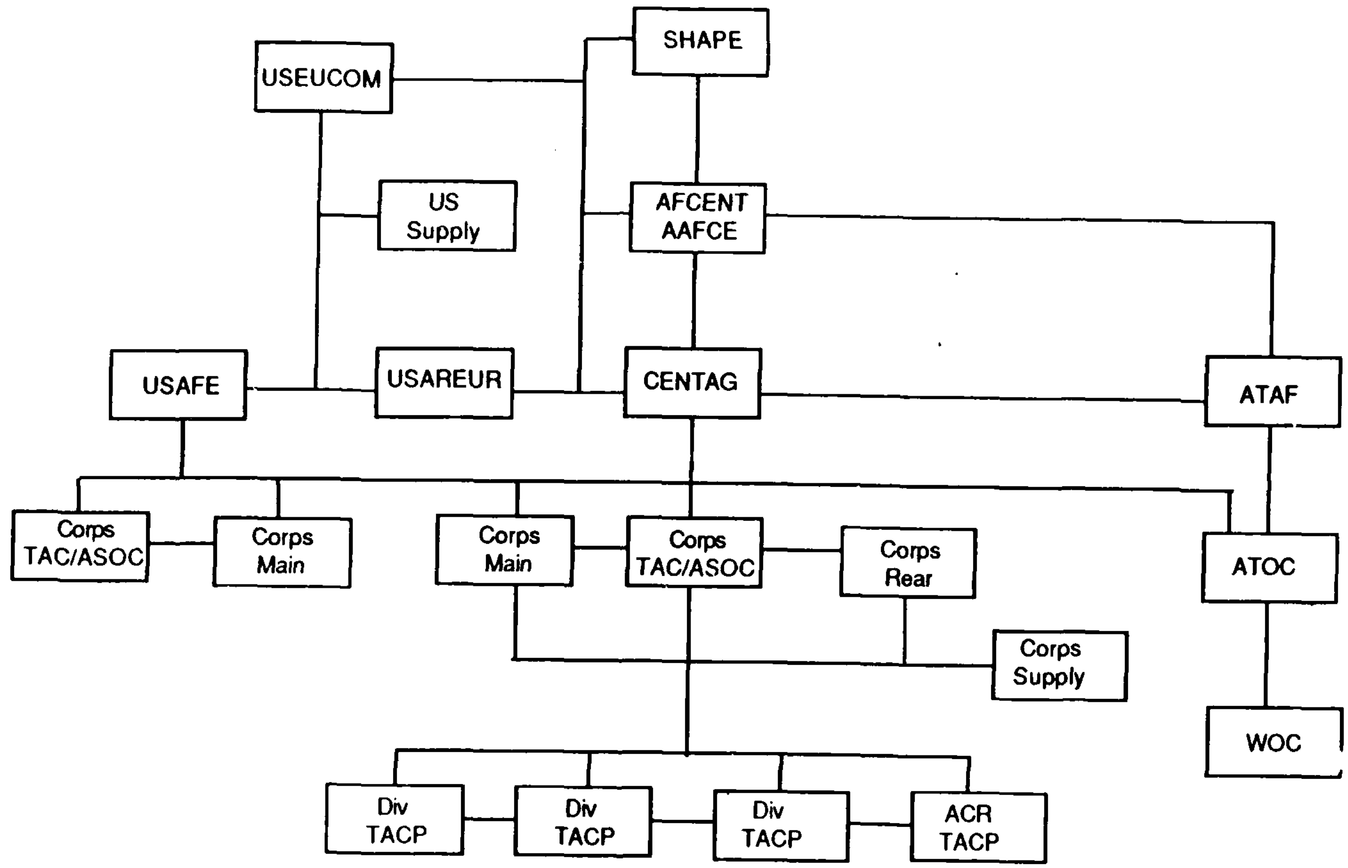


Figure III-1. C3 NETWORK

This "dimensionless" code is further enhanced by the use of linked lists of data structures in addition to the standard common arrays. The utilities section of this manual describes the DM subroutines and those that assist in the use of linked lists. This code uses explicit dimensions for the number of different combat systems that will be evaluated. This size was set to 13 distinct combat systems for the current version.

The three functional modules controlled in C3EVAL as shown in Figure III-2 are entered through Subroutines INPUT, EVENTS and OUTPUT. The C3EVAL process starts with determination of the mode. (Initial run starting at time zero or restart run at TIMET.) Interpretation of input and output unit numbers and setting of DM to zero is accomplished for time zero runs. DM is zeroed by the utility Subroutine DMINT. For TIMET starts, Subroutine RESTOR is called to reset all dynamic memory and model parameters.

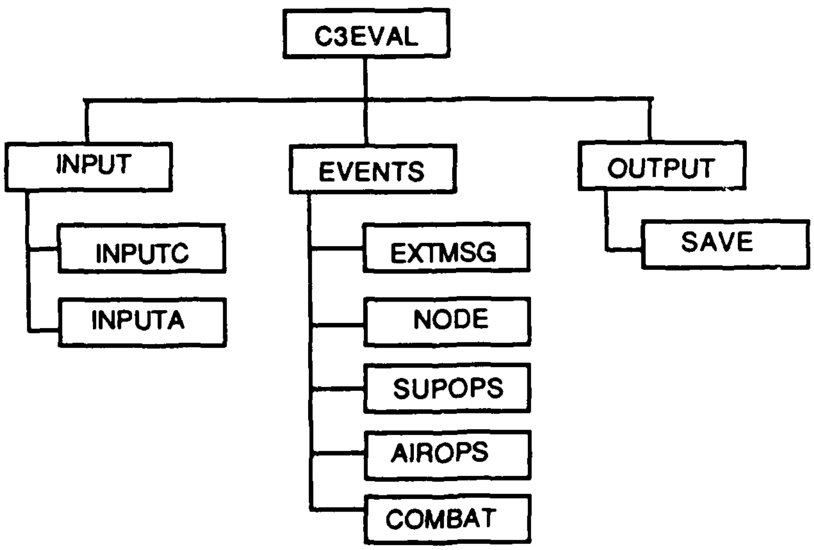


Figure III-2. C3EVAL MODULES

Subroutines INPUT and RULEIN are called for time zero runs to establish the decision rules and initial conditions for C3 and combat. Their operation is described in the next section. Subroutine EVENTS controls all C3 events, combat calculations and the game clock. After all requested simulated combat is calculated, summary output is produced by OUTPUT and C3EVAL

stops the run. The hierarchy of subroutines is shown in Figure III-3. This figure does not show the attrition module subroutines or the utilities.

A. CONTROL MODULE

The main program C3EVAL performs all the top-level functions of file interface determination and control of the mode of operation, data input, event simulation and output. The input files are:

C3DATA	C3 descriptions
CBDATA	Ground combat systems data
AODATA	Aircraft, helicopter and artillery operations data
C3RULE	Decision rule data
RANDIS	All distributions used by random calculations
RESTOR	All data necessary to continue a previous run.

The output files are:

C3ECHO	Echo of input data
C3TIME	All reports and error conditions
C3SUM	End of run summaries
SAVE	All data necessary to restart from end of a run
TIMET	Message flow data for each time increment
LOSST	Combat losses for each unit for each time increment
FORØ25	Optional debug output

Temporary work files are:

C3NODE	Changes to NODE data to occur during game
C3LMNO	Changes to node input and output limits to occur during game
C3LINK	Changes to link capacities to occur during game

1. Subroutine CONTRL

CONTRL is called by C3EVAL to read the print and debug parameters and to read the preamble from the C3DATA file. The preamble is used to document the contents of the file, to easily identify variations of a basic scenario and to assist in relating other files and output to a

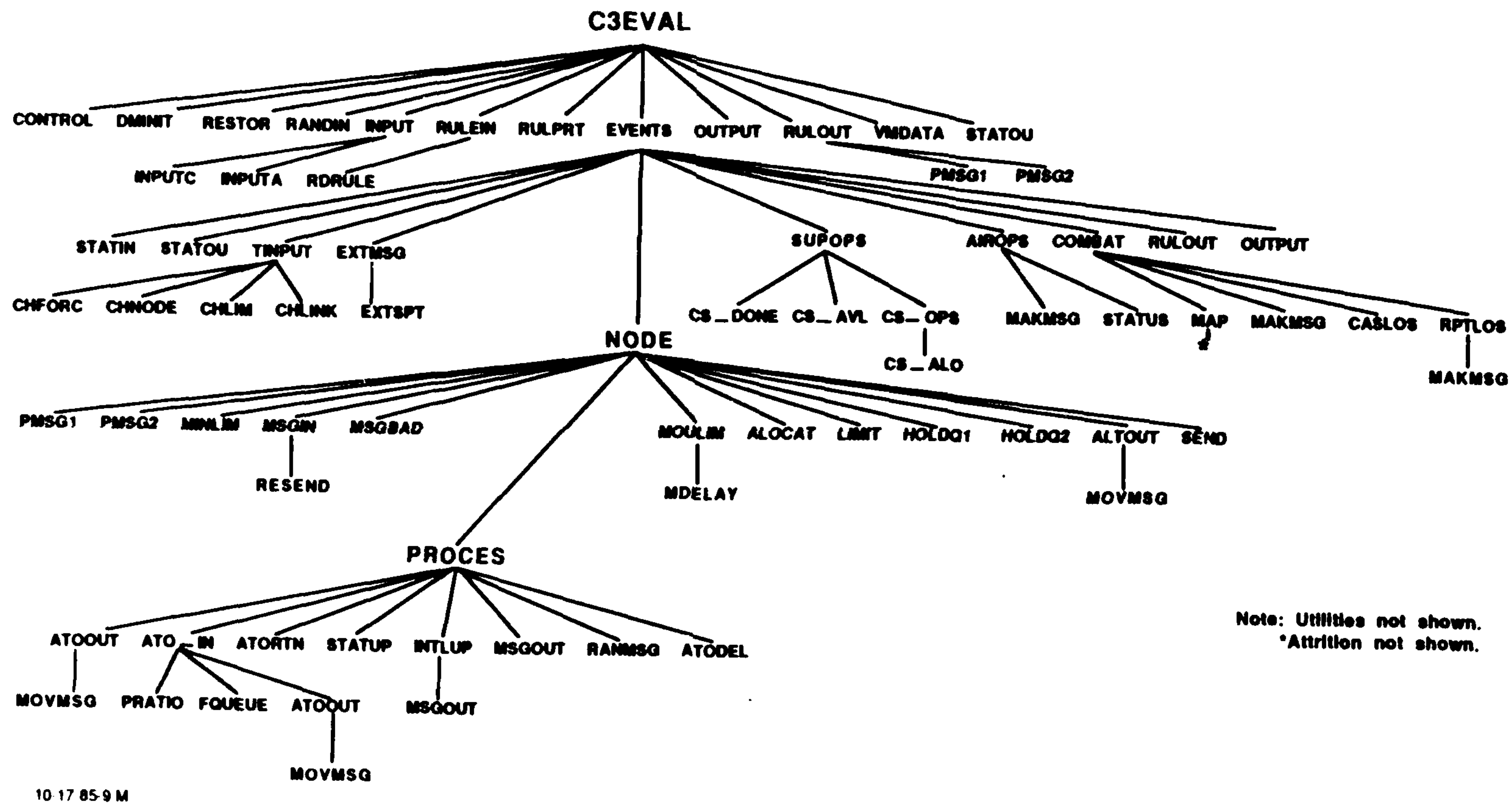


Figure III-3. SUBROUTINE HIERARCHY--C3EVAL

specific run. The first 40 characters of the top line of the preamble will appear as the subtitle on all graphics. The preamble may contain as many lines of 80 characters as necessary. It must be completed with "END " as the first four characters in the last line. CONTRL copies the preamble to files C3TIME, C3SUM and TIMET. The definitions of the flags are shown in Figure III-4 which is a PreProc screen. The flags are echoed to file C3ECHO.

```
PRINT FLAG (PFLAG(I)) DEFINITIONS
  1    ALL MESSAGES AT ALTERNATE DESTINATION
  2    ALL MESSAGES ON INPUT QUEUES
  3    ALL MESSAGES ON OUTPUT QUEUES
  4    ALL MESSAGES ON FUTURE QUEUES
  5    ALL MESSAGES BEING HELD
  6    ALL MESSAGES DELETED
  7    STATUS OF RULE STRUCTURE
  8    COMBAT SUPPORT SCHEDULED
9-10   NOT ASSIGNED
 11    TRACKED MESSAGES AT ALTERNATE DESTINATIONS
 12    TRACKED MESSAGES ON INPUT QUEUES
 13    TRACKED MESSAGES ON OUTPUT QUEUES
 14    TIME T OUTPUT ON FILE 14 REQUIRED
 15    COMBAT LOSS VECTORS
 16    FORCE RATIO CALCULATIONS
 17    RULE STATUS AT FINAL TIME
 18    AUTO POSTURE REQUIRED
 19    RANDOM PROCESSING REQUIRED
 20    USED INTERNALLY FOR SUM OF FLAGS
**********************************************
PRINT MODIFIER (PMOD(I)) DEFINITIONS
  1    OPTIONAL OUTPUT RESTRICTED TO THIS NODE
  2    OPTIONAL OUTPUT STARTS AT THIS TIME
  3    OPTIONAL OUTPUT STOPS AFTER THIS TIME
```

Figure III-4. PRINT AND DEBUG PARAMETERS

2. Subroutine DMINT

DMINT is called by C3EVAL when a game is to be started at time zero. It prepares DM for use by initializing the next available memory locator (MPTR) to one and setting all DM and the garbage pointer to zero. The garbage pointer is the root of the reusable memory block queue.

3. Subroutine RANDIN

Subroutine RANDIN inputs distribution types for changes to messages and all random distributions used in C3EVAL. It is called by Program C3EVAL and reads the input from file RANDIS.DAT as unit 16. The first line in the file is an 80-character comment line which is echoed to file C3ECHO. The distribution types of messages are read by format (A3, 6X, 3I6, 10A4). If the first three characters of the line are "LAS," message distribution input is terminated. The three

data fields are the message type number, the random distribution type for partial messages and the random distribution type for modification of message contents. Each line also provides for 40 characters of comments to document the data. The data is placed in an RDI block and linked to COMMON/RAND/PCDIS.

The second part of this input contains the random distributions. Each distribution set contains an integer and 11 real values. The data format is "free form" with variables separated by blanks or commas. The integer value is the distribution type number. The distribution values are for the uniform distribution draws at 0, .1, .2, .3, .4, .5, .6, .7, .8, .9 and 1.0. RANDIN reads the values, creates a 13 word data structure, RAND, for each distribution and enters the values in the structure. This data structure has its root in PRAND in COMMON/RAND/. There is no restriction to the number of different distribution types.

B. INPUT MODULE

Input data is of two types. The basic scenario data (or time zero data) is read by the INPUT subroutine and its subordinate routines (see Figure III-5). The EVENT (or TIMET data) is read by subroutines under control of EVENT (see Figure III-6). The Block Data FRATIO is included here because of its function. It is not called by INPUT of course.

1. Subroutine INPUT

Subroutine INPUT reads node, link and limits data in a manner that allows the user to build this file in alternative styles, depending on the form most applicable to the analysis process. Data is read from a file identified by the variable INP which is set to one in C3EVAL. The general form is to read a set name line, a header of 80 characters, and then a line of data. The first four characters are reserved to identify the type of input involved. All data lines must have the first four characters blank. This implies that this data line belongs to the set identified by the last set name read. Set name lines must have one of the following set names in the first four characters: NODE, LMNO, LINK or PROC in order to input that type of data. Anything other than blank as a set name will cause the input of the data set to end (normally set to LAST). This allows the analyst to put all node data together in file INP or to have a mixed sequence of set types and normally grouping NODE, LMNO, LINK and PROC data for each unit identified. A header line must always follow immediately after a set name line. It may be used as comments about the data or left blank.

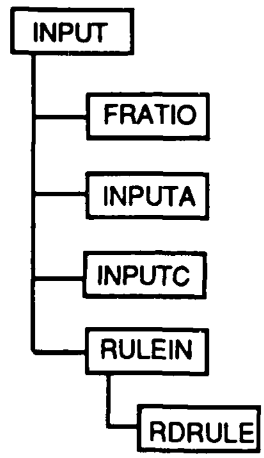


Figure III-5. SCENARIO INPUT SUBROUTINES

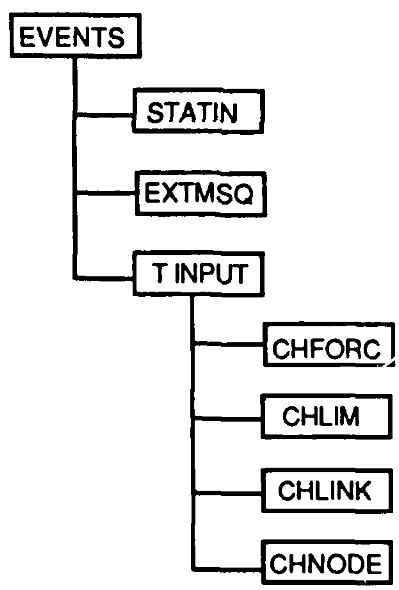


Figure III-6. TIME INPUT SUBROUTINES

In the program, the NODE set is located at the top of the subroutine. The first time a new node number is read a NODE data structure is created and unit number, identifier and type are set. The same node number will be required on each data line that identifies a different destination with which the node can communicate. Each time a new node number is found it is linked into the node queue which has its root in NODE1 in COMMON/C3/. A destination (DEST) data structure is created for each destination identified. the DEST data structures are linked together in the node's destination queue. The DEST data structure is initialized with its unit identifier and one or two alter destinations. The commander's element in the node data structure may be set to the destination's unit identifier. If a destination is a subordinate of the node, the subordinate flag is set in the node's DEST block. If the input value for the subordinate flag is equal to two then a subordinate status data structure is created and snapped into the node's subordinate status queue. Each node structure is also initialized with a message count COUNT block and a random distribution RND block.

The NODE data is read by format (A4, 8I5, 3A4, 24A1). The first field is the data set type (NODE on the first line and blank on all others). The eight numbers are: time, unit, unit type, destination, alternate node one, alternate node two, commander flag and subordinate flag. The flags indicate the relationship of the destination to the unit. The 3A4 field is the unit identifier and the remaining data are comments (normally the destination's identifier). If time is greater than one, the data line is written to file C3NODE to be processed at the future time. Note: Unit number 99 is required for the combat air patrol (CAP) function. This unit must exist in node data for random operation.

The next set in INPUT is LMNO, the LiMits for iNput and Output. It is read by format (A4, 9I5, 30A1). The first field is the data set type. The nine numbers are: time, unit, four message limits and distribution types for message length, message delays and request close air support (CAS) delay. If time is greater than one, the data line is written to file C3LMNO to be processed at the future time. The limits are: hold for input, delete for input, hold for output and delete for output. The comment field may be used to document the line (i.e., curtailed operation due to direct attack).

The next set in INPUT is LINK, which indicates the capacity of each type of communication between two nodes. The format used is (A4, 4I5, I10, I5, 41A1). The first field is the data set type. The six numbers are: time, two unit numbers, communications type, capacity and message lost distribution type. The last field is for comments. If time is greater than one, the line is written to file C3LINK. Notice that a link has a node at each end and that these nodes must

have been identified before any link information can be given. A particular link should be identified only once, and the input procedure for the LINK set is indifferent which node of the link is given first or second. A link line is required for each type of communications desired between each node pair. The LINK process finds the data structure for the node identified on the link input line. Then a LINK data structure is created and linked into the destination structure for the node. The type of link and its capacity are set in the link structure. This process is then repeated for the node on the other end of this link. The final data set recognized by INPUT is PROC. This type is not currently used and exists for compatibility to earlier versions.

When a non-set name is found by INPUT, it branches to input combat data. This is done by a call to INPUTC. Then air operations data is input via a call to INPUTA.

At this point all data has been read in, but there is still some initialization required for all node and destination structures. The first step is to replace the commanding unit identifier with a pointer to the commander's node data structure. Next, the unit type in all destination data structures are initialized by finding that data in the destination's node structure. Finally, the alternate/s are located by their unit number and the destination data elements for alternates are set to pointers to the alternate's node structure.

2. Subroutine INPUTA

This routine reads in air operation's data from file AODATA. This file has a documentation preamble which is followed by a comment line, CAS factors and allocation parameters. The factors are read by format (A3, 6X, 2I6). The first variable is not used. The second is the wait time required by combat-level units before requesting additional general support. It is stored in COMMON/TIME/NXTATO. The third factor is the distribution type for CAS sortie assignment. It is stored in the WOC structure at offset 11. The factor line is followed by a comment line and one or more lines containing aircraft allocation parameters. This line is read with format (A3, 6X, 6I6, 10A4). If the first field is blank, the data is processed; if it is equal to "LAS," the data set is terminated. The six numeric parameters are: allocating node, message number that the parameters are to apply, initial allocation time, number of periods to be used in the allocation process, the maximum number of sorties for a flight and the maximum sorties to be allocated throughout the specific periods. The last field is a comment field. INPUTA finds the node structure indicated, creates and initializes an allocation structure and snaps it into the nodes allocation queue.

The next data set in AODATA identifies the Control and Reporting Center (CRC) node. A documentation header is read. The first data line contains the CRC node number, the alert time, enroute time, the minimum number of aircraft that constitutes a CAS sortie and the probability of survival enroute of the aircraft on a CAS mission. The program creates the CRC structure and sets NCRC in COMMON/C3/ to this data structure. It stores the CRC data values in that structure. A WOC structure is created for aircraft availability data. A header for aircraft availability is read and data from the CRC data line and the first aircraft availability line is used to initialize the Wing Operations Center's (WOC) data structure. Each aircraft availability line is read by (A3, 4I6, 10A4). The four numeric parameters are aircraft base node number, aircraft type number, time aircraft are available for takeoff and number of aircraft available. The data structures created have a READY block for each aircraft type. The READY block has a time sorted queue of RDYQ blocks (one for each specific available time). The first RDYQ block is created empty (i.e., time = 0, number of aircraft available = 0) and is used during the simulation to hold the aircraft currently available. Input aircraft available data is put into subsequent RDYQ blocks.

After the aircraft data set is complete, a header for helicopter operations is read. The data line following it contains the helicopter base node number, the number of helicopters on a normal mission, enroute to target time, minimum helicopters on a mission and the probability of survivability of the helicopters enroute to the target. A SUP data structure is created and linked to the base's node structure. Then a HOC data structure is created, linked to the SUP data structure and initialized with the helicopter operations parameters. Next a READY/RDYQ is created as for the aircraft and each line of helicopter data is entered.

Finally this process is repeated for general support artillery operations and artillery tube availability under the SUP block.

3. Subroutine INPUTC

This routine reads in the combat weapon system values from CBDATA for each combat-level node. This file has a documentation preamble which is followed by the generic Red unit data set. Each data line contains a Red unit type, 13 values for the weapons systems, engagement rate type and the posture for this type unit. This data set is completed when unit type is equal to "99999." A Red table of equipment data structure is created for each line of data and snapped into the generic Red unit queue with root in variable NREDTE in COMMON/C3/.

The next line is a header line followed by a line with four combat support force ratio values. The values are the limit of the Red to Blue ratio that must be exceeded before a Blue combat unit may be allocated immediate CAS, helicopter or general support artillery. They are stored in COMMON/ENGMT/FRBCAS. This line is followed by a header line and four combat posture force ratio values. The values are the force ratio limits for Red and Blue posture control. Next is a header line and two lines of force ratio weighting factors. The first line has 13 weights for Red systems and the second is for Blue. The data is read by unformatted read and stored in the /RED/ and /BLUE/ COMMONs.

The last data set read by INPUTC is the combat weapon systems data. The first line read is a comment line to assist the analyst to identify the scenario data it represents. Each data line contains a node identifier engagement rate unit posture and number of weapons in each combat system type. It then finds the NODE data structure that is specified, creates a CMBT data structure for that node and stores the number of systems in that structure. When the NODE identifier is found to be "99999" the input is completed. The final section initializes the combat support force ratio index arrays (INDXFR and FRVAL). These arrays are sorted by force ratio limit value.

4. Subroutine RDRULE

RDRULE reads in the command post (node) rules which consist of three data sets. The data is read from file C3RULE. The file has a data preamble followed by three lines of heading for the rule parameter data set. This set is read by format (I5, I10, 4I5, 2X, 3A4). The variables read are: rule number, type unit that uses the rule, time required to perform the processes, the minimum messages required to initiate the process, an indicator of a periodic or reaction-type process and the start time for periodic processes. If the minimum messages required is zero, then the process is done for each current message. The last field read on each line is the title of the decision rule. This data set is completed when a rule number is zero.

The next data set read is the input message data. It has two lines of heading and is read by format (I5, I10, 3I5, 2X, 3A4). The variables read are: rule number, unit type that originated the message, message type, maximum age for the message to be useful and a flag to force a message to be used only once. This is to keep an input message that is retained for more than one period from generating the same output message more than once. The last field is the title of the message. The data set is completed when a rule number is zero.

The final data set is the output messages to be created by a rule. It has two lines of heading and is read by format (I5, I10, 2I5, 3(I5, I6), 2I2, 2I5, I2, 2X, 3A4). The variables read are: rule number, destination unit type, output message number, priority, three sets of link type and capacity required, flag if destination is commander only, output flag, two alternate destination unit types, the maximum time the message will remain active in the communications network and the title of the output message. Note: This data is read into 8 arrays in COMMON/RULE/. These arrays currently have a maximum of 400 entries. The output flag for random process types is also used as the minimum time between report generations.

5. Subroutine RULEIN

The subroutine creates the data structures OMP, IMT and MSG for each node. The parameters for the process, input messages and output messages are read from file C3RULE by Subroutine RDRULE. RULEIN processes each rule in array IRULE until IRULE (1,N) is equal to zero. For each rule number a queue of generic messages is created from the MSGOUT data from C3RULE. This queue has its root by rule number in the MSGQ array. After this queue has been created, an output message process (OMP) structure is created for each node of the type indicated as an originator of the process. This OMP structure will have a pointer to the queue of generic messages that will be created each time the process is successfully initiated during the game. This success is based on receiving the desired information at a node. This information is indicated in an input message table (IMT). The IMT is created by RULEIN from parameters from C3RULE and is attached to the OMP data structure. At the completion of initializing all rule processing, RULEIN calls Subroutine RULOUT to echo the data structures to output.

6. Subroutine STATIN

STATIN is a part of the input module because it initializes the data values in a commander's status data structure. This data is the perceptions by the commander of his subordinate unit's strengths and combat postures. The initial perceptions are based on the input numbers of weapon systems for tne subordinates. The perceptions of the subordinates for (Red) are set to the first generic Red unit that was read in and the foe unit title is set to "unknown unit."

STATIN is called by the EVENTS subroutine prior to entering the combat time loop. It initializes only the nodes that have status blocks crated by INPUT.

7. TIMET Input Sub-Module

C3EVAL has the ability to accept scenario-based messages created by the user and to modify combat units' force structures, command posts' input and output processing limits, communications paths capacities and preplanned changes in the command structure. These changes are indicated in input data by specifying the desired game time for them to take effect. Subroutines EXTMSG and TINPUT are called by Subroutine EVENTS during each game cycle. The other subroutines in this section, EXTSPT, CHFORC, CHLIM, CHLINK and CHNODE, are used to support these routines.

a. **Subroutine EXTMSG.** This subroutine reads in all external messages from C3DATA. These messages must be in time-sorted order. The time that a message is to be used is read into MIN(1). If this is some future time, processing is returned to EVENTS. When messages are to be processed, a MSG block is obtained from DM and filled with the input elements. If an ATO is indicated in the message, a DATA block is obtained, the ATO data read and transferred to the block. The ATO is attached to the message via the pointer PDATA. Next the node that is to receive the message is found and the message is put on the node's input message queue. The format for messages is (9I6). The variables read are: message times, message type, unit type of originator of the message, destination unit, unused field, output flag, additional data flag, priority and time message was created. The format for additional data is determined by the message type. If the message type equals 3136, the data is read by Subroutine EXTSPT. Otherwise it is read by format (6X, 6I6) and the variables are: support unit, earliest support time, latest support time, type of aircraft and number of aircraft.

b. **Subroutine EXTSPT.** EXTSPT reads additional data for message type 3136 from C3EVAL. The format used is (6X, 4I6, 5X, 3A4, I1). The variables read are: unit, time, type of foe, posture of foe, foe's title and flag to indicate additional data on the next line. EXTSPT creates an Intel Report Data Block for each data line and snaps the queue into the additional data pointer in the message.

c. **Subroutine TINPUT.** Subroutine TINPUT inputs changes to characteristics during a specified TIMET. The characteristics that can be changed are: number of weapons, engagement rate and posture of a unit, input and output message limits at a particular node, a node's commander or subordinate and the message capacity of a particular link type between two specific

nodes. The reinforcement changes are read from a formatted file. The node, limit and link changes are read from three unformatted files. Future changes are input and held until the specified future clock time. There may be more than one change per characteristics at any given clock time. The input stream must be in time ordered sequence. When an update time occurs, the appropriate subroutine is called to process the change.

d. Subroutine CHFORC. Subroutine CHFORC changes the combat values for a specified unit number for a Blue combat unit and its corresponding Red combat unit. If the unit number of the Blue combat unit does not exist (not found in NODE queue) then an error message is produced. Otherwise, the engagement rate and the posture of the Red and Blue combat units are updated. The number of Red and Blue weapons are updated for reinforcements. Note: A blank line must be given for a side that has no change.

e. Subroutine CHLIM. Subroutine CHLIM changes the input or output message limit at a specific node and random distribution types. The message limits are: maximum number of input messages that can be received at a particular node during one time increment, maximum number of input messages that can be received or held at a particular node, maximum number of output messages that can be sent from a particular node during one time increment, maximum number of output messages that can be sent or held at a particular node. The distribution types are: message length, message delay and request CAS delay. Note: All values must be given even if one does not change.

f. Subroutine CHLINK. Subroutine CHLINK changes the maximum capacity of that particular link type between two specific nodes. The values passed to CHLINK are: clock time change is to be made, unit identifier of node at one end of link, unit identifier of node at other end of link, link type and new maximum path capacity

g. Subroutine CHNODE. Not implemented yet.

h. Subroutine Block Data. This routine contains the data used by the force ratio calculation sub-module. This data includes the names of the weapons system types, engagement rates and combat postures. There may be three sets of engagement rates and postures for Red and Blue. Variable I1 in COMMON/BLUE identifies the index weapon systems and the number of

weapon systems is set. The allocation matrix for a generic unit is set for Red and Blue. Finally, the Red and Blue probability of kill matrices are set.

C. EVENTS MODULE

The EVENTS module contains all of the C3 combat, air operations and TIMET INPUT/OUTPUT. This section will discuss the functional subroutines in this module. The utility subroutines that are used in this module will be described. The structure of the event processes is shown in Figure III-7.

1. Subroutine EVENTS

The EVENTS subroutine is an executive routine that controls the game clock and sequence of events. For each time period external events are obtained by TINPUT and external messages are obtained by a call to EXTMSG (see Section B). Messages are received at command posts, processed and new messages generated by the call to NODE. Requests for general support (CAS, corps helicopters and artillery) are processed by SUPOPS. CAS sorties are scheduled, controlled while airborne, landed and rescheduled for COMBAT. Game time is incremented until the end of game time is reached, when processing is returned to C3EVAL Subroutine EVENTS checks the user's indicator for TIMET output data. If this is requested, Subroutine OUTPUT is called at each time interval.

2. C3 Sub-Module

This sub-module represents the command post actions (as specified by the decision rules) and the communications between command posts. Communications are represented by data specified paths between nodes and the capacity of each path to carry message traffic based on priorities. Messages can be generated by user (external) input, by random occurrence, based on decision rule parameters or in response to internal events such as receipt of a message or the change in a force ratio beyond input limits. Messages may be sent, delayed or deleted from the network. Each path capacity may be modified at any time interval to represent direct attacks, electronic warfare, etc. Each node has input and output limits on the number of messages that can be processed during each time cycle. Designated nodes will maintain perceptions of their subordinate's capabilities and the opposing forces via messages received. Allocation of support weapons are based on rules applied to these perceptions. The communications network will

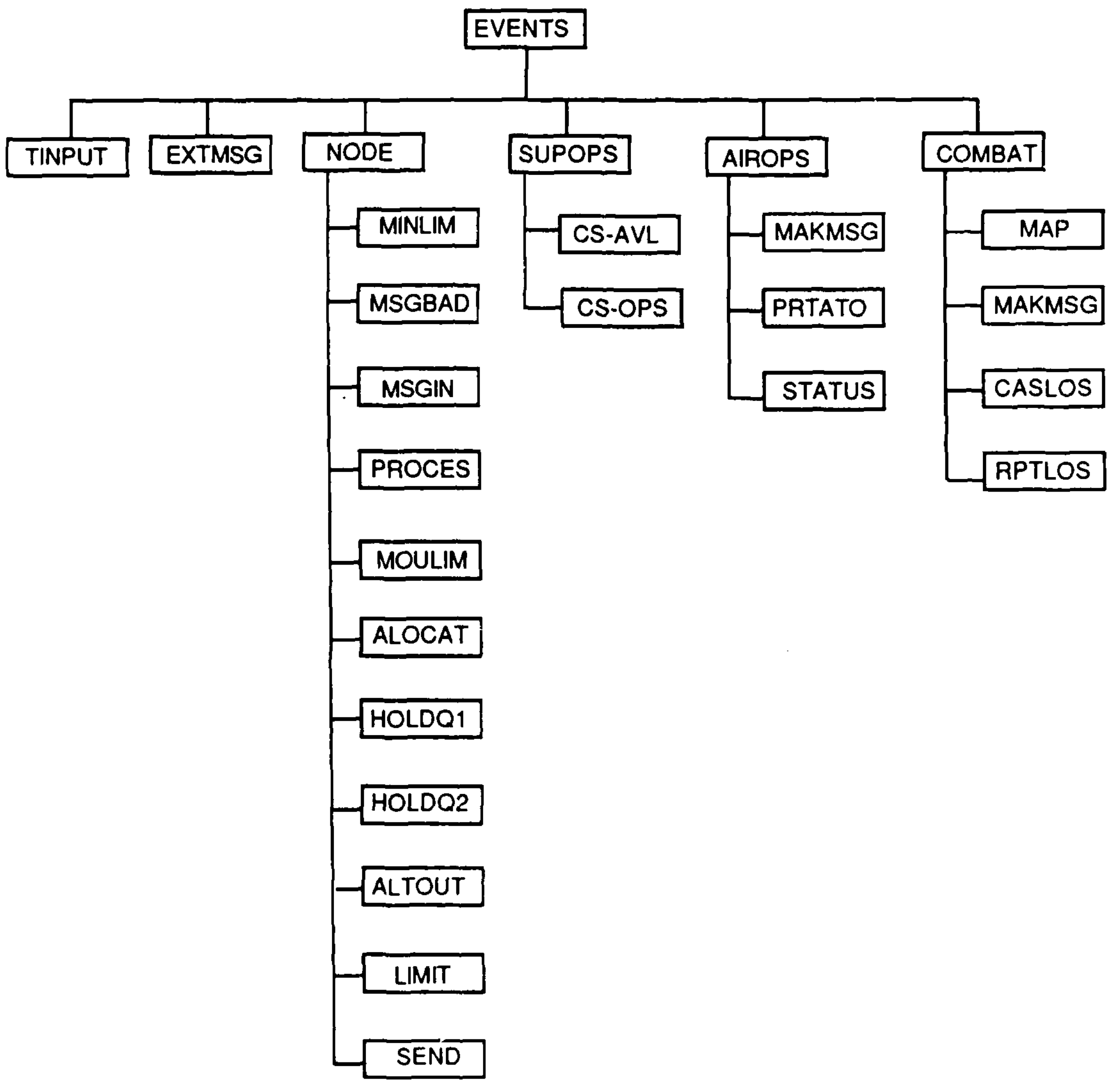


Figure III-7. STRUCTURE OF THE EVENT PROCESSES

attempt alternative communications types and routings to send messages that may be delayed. The details of message creation, movement, arrival and destruction are made available in continuous and summary form. All C3 processes are accomplished by Subroutine NODE and the routines that support it.

a. Subroutine NODE. This subroutine has three main sections to model a command post's C3 events. The first section processes the input messages that are on its input message queue. In this section each message is counted and checked to see if it is addressed to the node or has been sent to the node to be routed to its final destination. Rerouted messages are simply moved to the output hold queue for communication handling. If the destination for the message is not on the node's destination queue, the misrouted message is deleted and an indication of this action is put in standard output. All input messages at each node are reviewed by Subroutine MINLIM, where input limits are applied. MINLIM is also called after processing to put the messages held due to limit on the node's input queue for the next time cycle. Subroutine MSGBAD determines if a message is readable based on a random distribution. Subroutine MSGIN is called to process the messages that meet the limits.

When all incoming messages have been processed, the decision rules section is entered. This section models all processes that have been specified by input for this node type. A process may be periodic or based on reactions to input messages. If it is periodic and the time for the process is the current time, Subroutine PROCES is called to generate the output messages. Certain processes have been reserved for random creation of their messages. They are identified by process numbers 3800, 4800, 5800, 5900, 6800, 7800 and 7900. These processes are handled as periodic processes in Subroutine NODE. If a process is reactive, the input messages by type, originator type and number are checked to see if sufficient current information is available to complete the process. If the process is able to be completed, then Subroutine PROCES is called to generate output messages where rules have been met. After all processes have been completed at a node, Subroutine MOULIM is called to limit the number of messages that will be output to the network.

After all processes have been completed for all nodes, the communications section is entered. This process is modeled by a series of subroutine calls that review various aspects of allocating messages to available communications links capacities. The sequence of subroutine calls for message allocation is shown in Figure III-8. In that figure Subroutine LIMIT is shown after

each allocation routine because it calculates the effects of the two-way communications limit, while the other routines use the simpler one-way limit. The first acceptable communications link found is used by each routine. Therefore, to model this complex process each destination and link must be checked separately. In addition, messages may be bumped by higher priority messages but may have sufficient priority to bump other messages on different destination/link combinations. Therefore, each process is tried twice. At the end of this sequence all messages that are successfully communicated are moved to the receiving node by Subroutine SEND and process control is returned to Subroutine EVENTS.

```
•PRIMARY DESTINATION
   ALOCAT, LIMIT                             by primary links
   •FIRST PASS FOR ALTERNATE LINKS
      HOLDQ1, LIMIT                          1st alternate link
      HOLDQ2, LIMIT                          2nd alternate link
   •SECOND PASS FOR ALTERNATE LINKS
      HOLDQ1, LIMIT                          1st alternate link
      HOLDQ2, LIMIT                          2nd alternate link
•ALTERNATE DESTINATIONS
   ALTOUT, LIMIT                             1st pass, all links
   ALTOUT, LIMIT                             2nd pass, all links
•FIRST ALTERNATE DESTINATION
      HOLDQ1, LIMIT                          1st alternate link
      HOLDQ2, LIMIT                          2nd alternate link
•SECOND ALTERNATE DESTINATION
      HOLDQ1, LIMIT                          1st alternate link
      HOLDQ2, LIMIT                          2nd alternate link
•RECHECK ALL POSSIBILITIES
   ALTOUT, LIMIT
•SEND MESSAGES
   SEND
```

Figure III-8. MESSAGE ALLOCATION SEQUENCE

b. Subroutine ALOCAS. This routine allocates sorties to requests for CAS. It is called ATOALO, which establishes the number of sorties which can be allocated in accordance with the existing plan and prioritizes the requests received. ALOCAS approves or disapproves the requests based on the number of sorties specified by ATOALO. Approved requests are forwarded as messages to the Wing Operations Center. Disapproval messages are sent to the requester. The count of sorties approved under the plan is updated.

c. Subroutine ALOCAT. This subroutine is used by Subroutine NODE to initiate the allocation of message traffic to the elements of the communications network. Each message on the future message queue that is to be sent during the current time slice is moved to the appropriate destination link's hold queue in priority order. The network life of each message is checked. If it has expired, the message is deleted and the count of output messages killed is incremented. When this is complete, the hold queue is moved to the send queue and the send queue is checked to see if there is more message capacity required than the link can carry during the time slice. Any messages that are over the link's capacity are moved back to the hold queue. (Note: This is a one-way check. This means that this part of the allocation algorithm assumes that this node could use all available communications capacity when in fact there is another node at the other end of the link with messages to send as well. This condition is adjusted in Subroutine LIMIT, which is called by NODE immediately after ALOCAT is finished.

d. Subroutine ALTOUT. Allocation of message traffic in a busy network can be an involved process. The algorithm used by C3EVAL is intended to model the actions that would take place in a message center. It follows capacity limits of the network's links and message constraints of priority, link types and acceptable routings. The number of parameters that are involved are indicated in Figure III-9. The message may have nine possibilities (three destinations x three link types). The node may have several destinations each with its own set of link types. The primary destination link is tried by Subroutine ALOCAT. The HOLDQ subroutines allocate to a specified destination using either alternate link one or two. Subroutine ALTOUT allocates by any acceptable link (including the primary link) to message and node alternate pairs. This is accomplished by four successive calls to the utility Subroutine MOVMSG. ALTOUT sets the nodes two-way allocation limit flag off because this limit may be exceeded by its process.

e. Subroutine ATO_IN. Called by PROCES for each node that is processing requests for general support (message type 3400) and has a non-null pointer to its allocation parameters queue. ATO_IN processes all pending requests with one call for each message type. When the first request is received for a unit, ATO_IN creates an entry in an allocation queue. Any additional requests that are active for the same unit are consolidated into this entry. After all appropriate requests have been consolidated, routine PRATIO is called to calculate the processing nodes perceived force ratios for all the subordinate units. Then FQUEUE is called to create a separate set of pointers in the subordinate's status structure sorted by perceived force ratios.

Node		Message
DEST		DEST
PLINK	LINK1	ALT1
	LINK2	ALT2
	.	LINKP
	.	LINKA1
	.	LINKA2
	LINKn	
ALT1		
PLINK	LINK1	
	LINK2	
	.	
	.	
	.	
	LINKn	
ALT2		
PLING	LINK1	
	LINK2	
	.	
	.	
	.	
	LINKn	

Figure III-9. PARAMETERS IN COMMUNICATIONS ALLOCATION

f. **Subroutine ATODEL.** Routine PROCES calls ATODEL at the completion of each process to delete all ATO requests and reply blocks that belong to the process.

g. **Subroutine ATOOUT.** This routine is called by PROCES and ATOALO to process requests for CAS without allocation restrictions that are carried by message types 2900, 3000 and 3400. ATOOUT checks each entry on the nodes ATO queue to find the ones that match the current process numbers. Routine MSGOUT is called for each acceptable ATO and the number of sorties approved for the requesting unit is increased in the node's status block.

h. **Subroutine ATORTN.** ATORTN is called by PROCES to pass along, in accordance with the decision rules, notification of receipt of requests for CAS. ATORTN checks each entry on the node's ATO return queue to find the ones that match the current process number. Routing for the message is obtained from the ATO's return destination list. A message structure is created and filled in accordance with the generic output message format for the rule. Alternate node

and communications data are set for the unique parameters of the node and the message is put on the nodes' send queue or on its future message queue based on time to create the message

i. Subroutine FQUEUE. Routine ATOALO calls FQUEUE to create a threaded list through a node's status structure. The order of the list is based on the force ratio in each status structure. FQUEUE first sets all pointers in this list to the "unset" value of five. Then it traverses the node's status queue to find the maximum force ratio in blocks with an unset value. When the maximum is found it is snapped into the force ratio queue which removes the "unset" value. This process is repeated until all status structures are sorted.

j. Subroutine HOLDQ1. The communications section of Subroutine NODE uses the HOLDQ1 and HOLDQ2 subroutines to move messages from a hold queue to an alternate communications link if the message has a sufficiently high priority. Each message has the capability of having a primary and two alternate communications type links and a primary and two alternate destinations. The alternate links and their capacities are specified in the "generic messages data structure" in COMMON/RULE/. This specification is necessary because the acceptable communications type is a function of message type and the capacity required is a function of link type and message type. Destination type is also a function of message type. However, the explicit destination node identify is a function of the message originating node. Therefore, COMMON/RULE/ contains unit type for destinations and the specific node identities are filled in by Subroutine MSGOUT when a message is automatically created. User input messages are placed directly into the node's input message queue by Subroutine EXTMSG and alternate links and node data is not used for this type of message.

HOLDQ1 has the function of placing each message on the hold queue onto a link that matches the type specified as the first alternate type link. It has the capability of using either the primary or one of the alternate destinations. This selection of destinations is made in NODE and passed to HOLDQ1 by the calling sequence parameter IPASS. IPASS equal to zero means the primary destination should be used. When IPASS is equal to one or two, then the corresponding alternate destination is used. HOLDQ1 also bumps all messages of lower priority that exceed the "one-way" link capacity as a result of moving a message to its alternate communications link. Note that "bumped" messages are returned to the hold queue for the message's primary destination and primary link.

The sequence of operations of HOLDQ1 is as follows:

- Set allocation flag for this node to off. This flag indicates that a two-way limit has been completed for this node. HOLDQ1 will modify the results of any previous two-way limiting and therefore this action will have to be repeated. See Subroutine LIMIT for a description of this flag's implications.
- Identify the appropriate destination based on IPASS value.
- Find nodes link with type equal to alternate link one of message (if it exists).
- If message priority is higher than a message on the send queue and capacity exists for this message, then move message to the appropriate (priority order) position in the queue.
- Set alternate communications flag to one.
- Move any lower priority messages on the send queue to their hold queue if they exceed the "one-way" capacity check.

k. **Subroutine HOLDQ2**. This subroutine operates the same as HOLDQ1, except that it uses the message's second alternative communications link for the desired communications type. See HOLDQ1 for description of operation.

l. **Subroutine INTLUP**. Routine PROCES calls INTLUP when the message type is 3136, intelligence update. INTLUP updates a commander's perception of a subordinate's combat foe. INTLUP checks each entry on the commander's spot intel queue to find the ones that match the current process number. The perception is updated if the spot report has data that is later than the current perceptive data. If the spot report is most current, all previous perceptions of foes are deleted and the current report of one or more foes are entered in the commander's status structure. The titles of the foe units are for specific units. The unit type will be used to refer to generic unit data for foe unit strengths.

m. **Subroutine LIMIT**. This subroutine is used after each of the allocation Subroutines ALOCAT, HOLDQ1, HOLDQ2 and ALTOUT. Its function is to insure that the two-way limit of a communications link is not exceeded in the allocation process. LIMIT does this by starting with the root node and checking the link limit for each link for each destination node. In order to be efficient when the root node and each subsequent node has been processed completely, a flag is set

to one in each destination data structure. When the key node (the node which will have all of its destinations processed next) starts to process a destination, the allocation flag indicates that the two-way check has already been accomplished with the destination. For example, if the root node is number one and the next nodes are two and three in the node queue, the process starts with one as the key node. LIMIT checks each link between one and two and sets the allocation flag. Then it processes one and three and sets that flag. The next step is to make two the key node. It is not necessary to process two to one, because this was done previously. Therefore, in this example the links between two and three would be checked and the process would be finished.

The capacity check is accomplished in message priority order by comparing the next message at the key node to the next message at the other end of the link. When the capacity is exceeded by a message, all of the remaining messages on both nodes' send queues are moved (in priority order) to the messages' "home" hold queue. This queue is the hold queue for the actual destination (not alternate) and on the primary link. The final step is to set the allocation flags in both destinations data structure.

n. Subroutine MAKMSG. This subroutine creates messages relating to ATOs. This type message differs from most message types in that it has an additional data structure (ATO) attached to the standard message structure via the pointer PDATA. The message priority is set at one and the maximum time for each message to be on the network is three time increments. If a random processing is requested, processing time to create the message and the length of the message (communications capacity required) are modified by random distributions. The delay time factor is based on the distribution type (MDEL) in the node's RND structure. The delay is added to the normal processing time. The length factor is based on the distribution type (MLEN) in the node's RND structure. The normal message length is multiplied by the random factor.

o. Subroutine MDELAY. Subroutine MDELAY is used by subroutine MOULIM to determine which messages will be sent, held or deleted. Alternate and future messages are ignored. MDELAY receives from MOULIM the value of the maximum priority level to be sent (JP1) and the number of messages to be sent from that priority level (JCI). MDELAY also receives the value of the minimum priority level to be delayed (JP2) and the number of messages not to be deleted from that priority level (JC2). All messages of priority level less than the value of JP1 are sent (a value of one being the highest priority). At the JP1 priority level, MC1 messages will be sent and the rest will be held. All messages at levels greater than JP1 and less than JP2 will

be held. At the JP2 priority level, JC2 messages will be held and the rest will be deleted. All messages above level JP2 will be deleted.

p. Subroutine MINLIM. Subroutine MINLIM limits the number of messages that can be input in one time increment. If there are more input messages than can be processed in one time increment, then excess input messages are temporarily moved from the input message queue to the hold queue. After the messages that are still in the input message queue have been processed, Subroutine MINLIM is called again and the input messages are moved back from the hold queue to the input message queue so they can be processed next time. If the maximum number of input messages that can be processed cuts off the input message queue in the middle of a priority level, then all messages of that priority level are received.

q. Subroutine MOULIM. Subroutine MOULIM limits the generation of output messages at each node based on the number of messages created to be processed at each node and message priority. If there are more messages than can be output in one time increment, then the appropriate number of messages (starting with the lowest priority) are either held (delayed one time increment) or deleted

r. Subroutine MOVMSG. This utility is used by Subroutine ALTOUT to attempt to allocate messages to alternative destinations. MOVMSG will attempt to move messages in a queue with its root at PMSGO (in a link structure) and the first message located at PMSG in DM. It will try the first or second alternate destination in each message based on the value of IALT. The unit identified in IUNIT is checked to see if this is an acceptable alternate for a message. If it is acceptable, then the links that are available to go to this destination are checked to see if they are acceptable links for the message. For a link that is acceptable, the message is placed on that link if it has sufficient priority over existing messages already on the send queue and if it does not exceed the available link capacity. The alternate communications flag is set in the message to zero for primary link type and one or two for alternate link types.

s. Subroutine MSGBAD. Subroutine MSGBAD is called by NODE, if random processing has been selected, to process changes to each message received at each node. The changes created in MSGBAD are the impact of receipt of a partial message and changes to the content of the message. The distribution type for partial messages is found by comparing the type

of the incoming message to the types in the RDI structures with root in COMMON/RAND/PCDIS. If there is no distribution for the message type, no action is taken. If an RDI match is found, a random factor is created by RANDOM. If the factor is less than one, there is no effect by the partial distribution. If the factor is equal to one, the sender is identifiable and a request for the message to be resent is created. This internal-to-the-code message has type number 9991 and MAKMSG is used to create the repeat message request. The request repeat message has its PDATA pointer set to the original message (which has been removed from the node's input stream) for use by the originating node to resent the message (see RESEND). NOTE: It is suggested that random distributions for type 9991 messages NOT be used, as this could create an endless cycle of requesting messages to be repeated. If the random factor is greater than one, the message is deleted.

Messages that have a random factor less than one may have their contents modified randomly. This effect is possible for message types 3000, 3400, 7000, 3126 and 3130. The distribution type for change of content is also found in the RDI structure. The number of aircraft is multiplied by the random factor for CAS requests. The reported losses are multiplied by the random factor for combat reports.

t. ***Subroutine MSGIN.*** *Each node in the network has the capability to process* messages. The data structure that holds the information about this process is the "output message process." The basic assumption for modeling the processing of messages that have been received is that received messages can be grouped together by message type and message originator unit type. MSGIN compares the input message to the message type and originating unit type for each message processed at the node. When matches are found, the process' "input message type" substructure, the "input message list" (IML), is searched for the specific unit identifier. If it is not found, an IML structure is created for the new unit identifier. In either case (existent or previous non-existent IML) the time the message was created is compared to the time of message creation in the IML. If the incoming message is newer, the message flag is set to one and message age is set to zero in the IML. In addition, the IML message creation time is updated

After all processes have been updated by the incoming message, it is tested to see if it contains an ATO substructure. If it does, the ATO is moved to the nodes ATO queue. Finally the data block containing the message is released to DM for reuse and control is returned to Subroutine NODE.

u. Subroutine MSGOUT. Subroutine PROCES calls MSGOUT for each instance that a node has met the requirements to satisfy a message process. The functions performed are a determination of the desired destinations, alternate destinations and message structure creation. The data structure used is OMP and its queue of output messages.

Each output message in the OMP is checked to see if it is addressed to the commander or the node if it is an air request. If the commander flag in the message is on (=1), the message is sent to the commander only. If it is not commander only or an air request, the message is sent to all units (nodes) that can be communicated with directly (in the DEST queue) that match the destination unit type found in the fifth word of the output message.

Next a message data block is located in DM and filled in with the data in the output message queue. The process for alternate destinations matches the unit types specified in the output message to the designated alternate units for the destination of the message. Finally, the message is placed on the node's send queue, by priority, or on its future message queue, by time to be sent.

MSGOUT also has the ability to randomly vary the amount of communications capacity required to send a message and the length of time required to create it. If random processing is requested, processing time to create the message and the length of the message (communications capacity required) are modified by random distributions. The delay time factor is based on the distribution type (MDEL) in the node's RND structure. The delay is added to the normal processing time. The length factor is based on the distribution type (MLEN) in the node's RND structure. The normal message length is multiplied by the random factor.

v. Subroutine PRATIO. Routine ATOALO calls PRATIO to calculate the commander's perceived force ratio for his subordinate units. Each of the 11 weapon system types for the foe is summarized over each of the foe units in the foe queue. The number of each system type is based on the generic unit strengths minus the number of foe weapons reported as destroyed in the status structure. The number of subordinate unit systems is taken from the status structure. Each weapon system ratio is calculated. The current unit force ratio for allocation of support uses the perceived strengths and the combat systems weighting factors. PRATIO also sets the force ratio calculation time to the current game time.

w. Subroutine PROCES. Routine NODE calls PROCES to perform response actions based on the conditions of process rules having been met. When NODE determines that the conditions have been met for a rule number, PROCES bases its actions on the output message types and the rule number. It currently has special processes for message types 2900, 3000, 3400, 7000, 9990, 9993, 3126 and 3136. The first step checks the rule number that initiated the action. If the rule is a random process, then Subroutine RANMSG is called. Random process numbers are: 3800, 4800, 5800, 6800, 7800, 5900 and 7900. If the rule is not a random process, it checks the first message type and branches to the related CASE statements. After each set of CASE statements, the next output message type from the rule queu is processed. When all output message types for the process have been completed Routine ATODEL is called to delete all additional data structures that were obtained from input messages under this process number.

x. Subroutine RANMSG. Routine PROCES calls RANMSG in response to output message type 3800, 4800, 5800, 5900, 6800, 7800 and 7900. RANMSG checks itime to determine if it is the start of the game. If it is, the node's random message queue is initialized with the processes random message using the start time in the generic message. Subsequent messages on the random message queue are checked to see if their send times have occurred. If they have, the messages are scheduled again using the generic time plus the random time and MSGOUT is called to create the actual messages to be sent.

y. Subroutine RESEND. Subroutine MSGIN calls RESEND to process a type 9991 message. RESEND takes the original message (found by using the pointers to PDATA in the RESEND message) and resets the message created time to the current time. It insures the pointer to the next message is cleared and moves the message onto the node's future queue.

z. Subroutine SEND. SEND is the final subroutine in message processing. It takes the contents of each link's send queue for each node and moves the entire queue as a complete string to the destination node's input queue. The resulting sequence on a node's input queue is by sending node and by priority of the messages within each sender's segment. During the counting of input messages, if random processing is selected, the distribution type for the link it is on is used to determine a random number. If the random number is not zero, the message was lost and the MSGONE counter is incremented.

3. Combat Support Operations Sub-Module

The combat support module receives requests for combat support from the combat level units. The arrival of requests at a node that has this capability causes the allocation of general support systems (CAS, helicopters and artillery) based on the preceptions of the subordinate's combat situation. Requests for immediate CAS are sent to the WOC. Helicopters and artillery resources are scheduled, placed into the combat matrices and withdrawn by this module. This module consists of five subroutines: SUPOPS, CS_DONE, CS_AVL, CS_OPS and CS_ALO.

a. Subroutine SUPOPS. This subroutine withdraws general support helicopter and artillery resources when they are better used elsewhere or are no longer required. This is done by calling CS_DONE for each node that has allocation parameters. The nodes queue of available helicopters and artillery systems are updated to reflect turnaround time by CS_AVL. The allocation process is done by CS_OPS. It also puts all newly assigned artillery tubes into the combat unit's matrix. SUPOPS then places the appropriate helicopter missions into the combat matrices and in takeoff status.

b. Subroutine CS_DONE. Withdraws all helicopters from combat matrix position 12 at each time cycle. Also withdraws all general support artillery from matrx position 13 when the support not required threshold is met. This calcualtion is based on the commander's perceptions of the combat situation. The withdrawn helicopters and artillery systems are placed into the available queues at the calculated times. The returning helicopters are reduced by the enroute attrition factor.

c. Subroutine CS_AVL. This subroutine makes the helicopters and artillery system availability queues current by collapsing the now available entries into the RDYQ at the top of the queue. The maximum available system limits are set. The current allocation parameters for CAS are retrieved from the node's allocation structure and the CAS sorties available for allocation are calculated.

d. Subroutine CS_OPS. Processes each request for combat support and deletes the request from the temporary request queue. The requests are sorted by subordinate's force ratio. Each request is checked to see if it meets the smallest force ratio limit. If it does, CS_ALO is called

to attempt to allocate the general support system with the smallest force ratio limit. If support was provided, the request's force ratio is checked at the next level. If support is not provided, the next higher system is tried without a limit check.

e. Subroutine CS_ALO. Allocates a general support combat system to a specified unit if the resources are available. The selection of its three cases (CAS, helicopters and artillery) is made by CS_OPS when it calls CS_ALO. If support is provided, the number of systems to be provided is calculated and subtracted from those currently available for assignment. The support provided flag is also set.

4. Air Operations Sub-Module

The air operations module receives requests for CAS from nodes representing ground unit command posts (usually at the corps level) in accordance with user established decisions rules. The reception of CAS requests and the resulting allocation, assignment and scheduling of aircraft to support the requests are modeled at the WOC. The structure of air force combat resources starts with a CRC that controls airborne CAS for a designated set of ground combat nodes. The aircraft that the CRC controls come from one or more notional airbases that have a direct relationship to the CRC. Each airbase (WOC) may have one or more types of aircraft that are scheduled for sorties. A queue exists for each aircraft type at each WOC. This queue contains the number of aircraft that will be available for assignment at a specified time. The aircraft combat cycle starts in the availability queue, includes assignment, takeoff, reporting into the CRC, enroute to target, combat attrition (in the supported ground units combat matrix), aircraft survivability, return to airbase, turn-around for another mission and back into the availability queue.

Requests for support are originated by the combat level units when their force ratios reach a user designated level. They may also be originated via EXTMSG input to a node. The requests for CAS are processed through the C3 network and command posts in the same way all messages are handled, with the exception that a commander may approve requests in accordance with an allocation plan. The WOC is a designated node type that has WOC processing capabilities and resources. Requests for CAS arrive at the NODE message structure for a WOC in the same way that all messages arrive at a node (in the node's input queue). Notification of action on a CAS request is returned in the same manner. This module consists of three subroutines: AIROPS, MAKMSG and STATUS.

a. Subroutine AIROPS. Aircraft resources on the ground at the notional airbase are maintained in aircraft-type queues as shown in Figure III-10. The first action by this subroutine is to determine the number of aircraft at the current game time. In Figure III-10, if time is three the program would add 3.93 to the previously available 4.0 aircraft and then delete the RDYQ blocks with time three. This is done for all aircraft types (READY blocks). The aircraft availability status is printed out if requested by user input. Mission takeoffs are scheduled by starting with the requested ATO queue in the WOC's node structure.

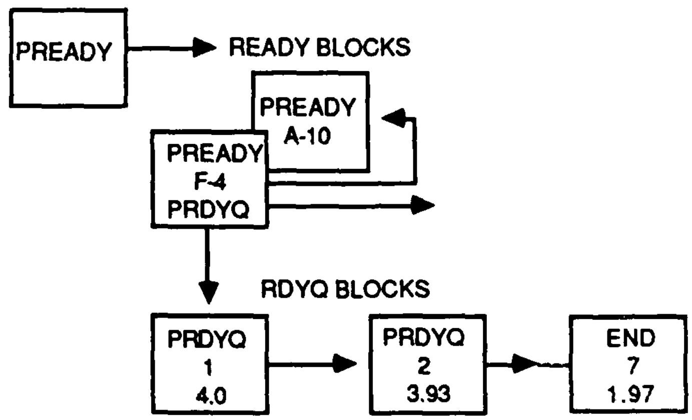


Figure III-10. AIRCRAFT AVAILABILITY STRUCTURE

The projected time on target (TOT) is calculated using alert time and enroute time and compared to the first request's earliest acceptable TOT. If the aircraft would arrive too soon, processing of air requests is finished for this WOC during this time frame. (Requests on this queue are sorted in earliest TOT order.) If the earliest time is acceptable, then the latest time is checked. If the mission would be too late, the request is deleted and a message is sent to the requesting node. If the mission can meet the requested window, processing is continued by checking the availability of the aircraft type requested and the number to be sent. All missions will have a whole number of aircraft assigned that is not less than the minimum aircraft limit specified by input. If the number requested is available, then the request is completely filled. If the number available is less than requested, the mission is scheduled with the reduced number. If random processing is selected and the random target distribution type in the WOC's structure is greater than zero, the mission will

have a random effect on its target assignment. If the random number is equal to zero, target assignment is normal. If not, the mission ends up on combat air patrol. However, if the random number is not one and another request exists, the mission goes to the other target. Note: CAP is designated as node number 99 and it must be in the C3DATA node data set.

If a request is fulfilled, messages to the CRC and requesting node are created by calling Subroutine MAKMSG and placed on the WOC node's future message queue. An ATO is placed on the CRC's ATO queue. This queue is used to model the mission reporting into the CRC after takeoff. After processing all current requests for the WOC, the remaining aircraft availability status is printed out. Subroutine AIROPS also processes CRC actions. The first step is to zero out all of the CAS entries in the combat matrix for each combat-level unit. (Note that this means that time on target is always one time cycle.) Then the ATO queue is examined for each mission that is on target during this time interval. The combat unit's combat matrix is found and CAS is incremented by the number of aircraft in the mission. Also losses of CAS aircraft due to enemy action are computed. A running summation of CAS sorties is also kept. The last step is to calculate the number of aircraft that survive the mission. An enroute survivability is factored in and the total returning aircraft is computed. Note that this may produce fractions of an aircraft. This number is then scheduled for landing and ground turn-around by entering it in the WOC's RDYQ queue.

b. Subroutine MAKMSG. This routine is called by Subroutine AIROPS to create message type 7000 that notify a combat unit of the time on target and number of aircraft that have been dispatched for CAS. This routine is documented in Section 3, 2, n above.

c. Subroutine STATUS. This routine is called by Subroutine AIROPS to print the status of a WOC onto file C3TIME.

5. Combat Sub-Module

This module uses the IDA method of attrition calculation documented in IDA paper P-1615, Net Assessment Methodologies and Critical Data Elements for Strategic and Theater Force Comparisons for Total Force Capability Assessment (TFCA), Volume II: Illustrative Example of Static Measures and Methodology. The executive routine for the attrition calculation is Subroutine MAP. It is called by Subroutine COMBAT which interfaces between the C3FVAL processing and data structures and the MAP algorithms. This sub-module generates reque ts for CAS, determines

aircraft losses during the attack portion of their mission, saves weapon system losses on file LOSST and creates spot loss reports.

a. Subroutine CASLOS. Subroutine AIROPS calculates the time on target and the time to return to the ready queue at the WOC for each CAS mission. It also calculates the enroute losses and schedules the remaining aircraft for return to duty by putting the returning aircraft on the WOC's ready queue. Attrition of CAS due to hostile systems in the ground support area is calculated by Subroutine COMBAT. COMBAT calls CASLOS to add the additional losses to the mission aircraft. CASLOS searches the WOC's ready queue for the first mission with return to ready time that is the same as the aircraft returning from combat, and subtracts the combat losses from the number of aircraft to be available.

b. Subroutine COMBAT. This routine is the executive for determining combat attrition. Combat interactions are evaluated each time increment for each unit that is a combat-type (has a CMBT structure) unit that has combat engagement of zero (i.e., in reserve). Combat system, engagement and posture data is extracted from a node's combat structure for both Blue and the opposing Red force. The allocation and kill potential matrices are calculated by Subroutine MAP. The Red-to-Blue force ratio is calculated from a linear combination of the existing weapon system strengths times an input weighting factor. The combat drawdown in calculated by the matrix multiplication of the number of systems times the opposing side's kill potential matrix created by MAP and the target system's postures. Postures are set as a function of force ratio by POSTURE. The results of the drawdown are stored in the node's combat structure and losses are output to file LOSST if the user has requested this data. File LOSST is used by the post processor to generate weapon system loss graphics. Subroutine COMBAT compares the current force ratio to the input threshold level and creates a request message for combat support if the ratio is too high and earlier requests are not pending. If the user requested it, the results of each combat engagement is printed on file C3TIME. If the user has indicated random processes desired, COMBAT will randomly modify the times that requests for support are sent. Subroutine COMBAT calls RPTLOS to generate spot loss reports to the combat unit's commander.

c. Subroutine MAKMSG. Subroutine COMBAT calls MAKMSG to create messages to request CAS support. This routine is documented in Section 2,n, above.

d. Subroutine MAP. This routine is the executive for calculation of weapon systems allocations and the Q combat matrix. This routine and its subordinates are documented as noted in 4), above.

e. Subroutine RPTLOS. This routine is called by COMBAT for each unit after combat attrition is calculated. It creates two combat spot loss reports, message number 3130 for Blue and 3126 for Red losses. These reports are sent to the unit's commander and form the basis for the commander's perceptions of the unit's status. The actual losses are reported unless the random process flag is on. Under random operations the messages may be randomly delayed by Subroutine MAKMSG and the contents of the messages may be randomly modified by Subroutine MSGBAD.

D. OUTPUT MODULE

Output by C3EVAL is provided in four different areas: input echo, game events, summary and TIMET. Input data is echoed to file C3ECHO to assist in creating a complete record of the scenario and to verify that the data structures created during initialization are properly filled. Subroutine INPUT echoes the data preamble, control flags, node data set, link data set and limits data set. Subroutine INPUT echoes combat unit strengths. Subroutine INPUTA echoes CAS allocation parameters, CRC parameters and available aircraft. Subroutine RULPRT prints out the decision rules in node type (echelon) order.

Game events are printed to file C3TIME. Subroutine STATOU prints out the commander's initial perceptions of his subordinates. TIMET input of reinforcements, combat status and link capacity changes are printed out by routines under TINPUT. EXTMSG echoes the user's input messages as they occur. The printout of message flows, queue status, combat losses and scheduling of CAS is controlled by user flags set by routine CONTRL. If the debug flag is set, a large volume of physical as well as logical data is printed on file FORØ25.

At the end of the game, a summary of communications, network and support sorties are printed to file C3SUM. This is followed by the size of dynamic memory actually used and the commander's perceptions at game end.

The fourth area of output is created for use by the graphic post processor. The message flow data at each time interval may be printed to file TIMET and the combat losses data printed to file LOSST.

1. Subroutine OUTPUT

Subroutine EVENTS calls OUTPUT if the user has requested TIMET data to be saved, PFLAG(14), for the post processor. EVENTS set the output file, IOUT to TIMET, calls OUTPUT, and then resets IOUT to C3TIME afterward. At this time the counters of random impact on messages are also printed. The main program C3EVAL calls OUTPUT at the end of the game with IOUT set to C3SUM. OUTPUT creates the printout shown in Figure III-11.

SUMMARY OUTPUT AT TIME 48

			COMMUNICATIONS LIMIT				INPUT LIMIT			OUTPUT LIMIT			SORTIES	
UNIT	NUMBER	TYPE	IN	OUT	HOLD	KILL	IN	HOLD	KILL	OUT	HOLD	KILL	CAS	HELO
CAP	99	9999	0	0	0	0	0	0	0	0	0	0	0	0
V CORP SUP	23	495	0	2	0	0	0	0	0	2	0	0	0	0
USAFE	22	6500	35	2	0	0	35	0	0	2	0	0	0	0
USEUCOM	21	650	3	0	0	0	3	0	0	0	0	0	0	0
US SUPPLY	20	525	1	3	0	0	1	0	0	3	0	0	0	0
USAREUR	19	550	1	0	0	0	1	0	0	0	0	0	0	0
SHAPE	18	700	50	15	0	0	50	0	0	15	0	0	0	0
AFCENT/AAFC	17	600	225	132	0	0	225	0	0	132	0	0	0	0
VII CORPS	16	400	18	8	0	0	18	0	0	8	0	0	0	0
VII CORP TA	15	450	37	16	0	0	37	0	0	16	0	0	0	0
CENTAG	14	500	101	61	0	0	101	0	0	61	0	0	0	0
V CORP REAR	13	490	53	38	0	0	53	0	0	38	0	0	0	0
V CORPS TAC	12	450	409	70	0	0	409	0	0	70	0	0	0	0
52 MECH	9	300	12	177	0	0	12	0	0	177	0	0	0	0
WOC	8	7000	5	115	0	0	5	0	0	115	0	0	0	0
ATOC	7	5000	117	279	0	0	117	0	0	279	0	0	0	0
4ATAF	6	6000	286	217	0	0	286	0	0	217	0	0	0	0
23 ARM DIV	4	300	16	178	0	0	16	0	0	178	0	0	0	0
V CORPS	3	400	186	66	0	0	186	0	0	66	0	0	0	0
20 MECH	2	300	15	136	0	0	15	0	0	136	0	0	0	0
201 ACR	1	250	20	80	0	0	20	0	0	80	0	0	0	0

Figure III-11. SUMMARY OUTPUT AT TIME T

2. Subroutine PMSG1

PMSG1 prints out data about messages at alternate destinations and on a node's input and future message queues. It may print all messages, messages at a specified node and/or during a specified time frame. It may be restricted to those messages that have their track flag set.

3. Subroutine PMSG2

This routine processes the same as PMSG1 except that it is done for a node's output and hold queue.

4. Subroutine RULOUT

Routine EVENTS calls RULOUT if PFLAG(7) is set. The main program C3EVAL calls RULOUT if PFLAG(17) is set. RULOUT prints out the decision rule parameters and variables in physical structure form so that the details of their operation can be followed. If the debug flag is on RULOUT, will call PMSG1 and PMSG2.

5. Subroutine RULPRT

The main program C3EVAL calls RULPRT to echo the decision rules to file C3ECHO. RULPRT loops through each decision rule for each level unit specified in its internal data statement. If new unit types are added to the scenario, the unit type must be added to this data statement to have its rules echoed to the file. The decision rules are printed in unit type order with the rule process parameters followed by the required message-in data and then the output messages to be generated by the rule.

6. Subroutine STATOU

Routine EVENTS calls STATOU at the start of event processing to print the commander's initial perceptions of his subordinates. The main program C3EVAL calls STATOU at the end of the game to print the commander's perceptions at the end. STATOU checks all nodes to determine if their status queue exists. If it does, the integer value of Blue and Red current perceived forces and losses are printed. Blue data and Red losses come directly from the status structure. Red perceived weapon systems numbers are obtained by adding the values for each foe perceived from the generic tables and subtracting the losses reported in the status structure. The perceived combat posture is also printed.

7. Graphics Data

Summary data is printed by Subroutine OUTPUT. If the user has indicated graphics post processor data is desired, Routine EVENTS calls OUTPUT at each time interval. This data is written in character format to file TIMET. The post processor reads this data to create some of its graphic output options. In addition, if the graphics data flag is on, Routine COMBAT writes the combat losses and force ratio for each node in combat. This data is written in binary form to file LOSST. The post processor also reads this file.

8. Subroutine VMDATA

This routine prints out the maximum dynamic memory location used and the status of the reusable block queues.

E. UTILITIES

These routines perform data structure building, searches, dynamic memory operations and other program support-type functions.

1. Subroutine FIND

Subroutine FIND searches the elements of a queue for an input calling sequence integer value (ID). The starting point in the queue is PIN. FIND assumes that the pointer to the next element in the queue is the first value in an element and that the last element in the queue will have this value set to zero. The offset from the first value of an element that contains the desired value is indicated by the parameter N. If the value is not found the output parameter POUT will be set to zero

2. Subroutine GIMME

GIMME provides a DM data block of length LEN from the memory space ISPACE. In the current version of C3EVAL there is only one dynamic memory area (MEMORY). The location of the first word in the data block is set into NPTR. GIMME first searches its garbage list to see if a block of equal length is available for reuse. If not, it creates a new block of the desired length by increasing next unused space pointer ISPTR by the value of LEN. GIMME also checks to insure that the current maximum size for DM is not exceeded.

3. Subroutine POUT

This routine is used for debug purposes only. It produces a snapshot of part of dynamic memory. The snapshot starts on location one and prints the specified number of variables (up to 11) per line and the specified number of lines. Note that C3EVAL does not use locations 1-100, and this area should be all zeros.

4. Subroutine RANDOM

Subroutine RANDOM locates the random distribution type indicated in its calling sequence and returns a random value based on the distribution. The queue of the distributions has its root in PRAND in COMMON/RAND/. This queue is searched for the indicated distribution type. If the type is not found, an error message is written to unit IOUT and the program execution is halted. If the distribution is located, a uniform random number (0.,1.0) is drawn by FORTRAN function RAN. The value returned is the interpolated value from the distribution that corresponds to the random number drawn.

5. Subroutine RELEAS

This routine works in conjunction with GIMME to control dynamic memory. It places data blocks on the reusable memory queue. The block address is placed on a queue of blocks that have the same length as the input parameter to RELEAS. The queue has its root in variable IGBPTR. Note that there is no garbage collection accomplished.

6. Subroutine RESTOR

Restore is used to restart a game at TIMET other than zero. It assumes that a previous run has been made and that the status of dynamic memory and COMMON parameters were saved by Routine SAVE. This capability has not been enhanced to operate with the current version of C3EVAL and will not operate correctly until it has the ability to handle the TIMET input files created in the current version.

7. Subroutine SAVE

This subroutine saves the status of DM and model parameters. This data may be used to restart the game at the point where Subroutine SAVE was called. Due to enhancements made to provide additional TIMET inputs, this subroutine requires enhancement before it can be used with the current version.

8. Subroutine SNAP

Subroutine SNAP finds the correct position in a queue with root in parameter PTR to insert a new member located at PIN. The sequence variable is located at NWORD in the data structure.

F. DATA STRUCTURES

There are two distinctly different approaches to data structures used in C3EVAL. The use of FORTRAN common variables and arrays is described first. The next section defines the approach used to provide essentially dimensionless code and the linked list data blocks that are used to implement the required data structures.

1. Common Data Structures

The following 12 named common data structures are used in C3EVAL. The first nine commons are contained in INCLUDE·FOR. This implementation is standard FORTRAN, with the exceptions of variables that start with "P" which are declared implicit integer variables. They are normally used to represent pointer (locators) of linked list data blocks in C3EVAL's dynamic memory.

a. COMMON/C3/NODE1,PGOMT,NCRC,INP,IOUT,NREDTE,IRAND

NODE1	Location of first node data structure in DM.
PGOMT	Not currently used.
NCRC	Location of first CRC data structure in dynamic memory.
INP	Input file number of for C3 data, set to one in Program C3EVAL.
IOUT	Output file number of all output, set to six, seven and eight in Program C3EVAL and 14 in EVENTS.
NREDTE	Location of first generic Red unit table of equipment.
IRAND	Seed for random number generator set to 731593 in Block Data FRATIO.
LOSSUM	Location of first CUML losses data structure.

The node data block is the basic building block for all C3 and combat data. The descriptions of the node, CRC and Red generic data blocks are in Section 2.

b. COMMON/SPACE/NOUSE,MEMORY(20000)

NOUSE — A check variable used for debug purposes only. (It is MEMORY location zero.)

MEMORY — DM array. It is equivalenced to STORE to facilitate its use for floating point as well as integer values.

c. COMMON/LOCATE/ISPTR,IGBPTR,MAXSP

ISPTR — Location of start of unused dynamic memory.

IGBPTR — Root of the linked list of resuable data blocks in DM.

MAXSP — Maximum value for ISPTR.

d. COMMON/TIME/ITIME,INCTIM,LASTT,PD1,PD2

ITIME — Current game time, number of INCTIM intervals that have been simulated.

INCTIM — Basic time interval of model, set to one in C3EVAL meaning one 30-minute poeriod.

LASTT — Last time for this simulation run (for 24 hours of combat LASTT=48).

PD1 — Start time for debug print (if PDEBUG=0).

PD2 — Stop time for debug print.

e. COMMON/RED/ and COMMON/BLUE/

```
      COMMON/RED/  NR(11),ALTCRB(11,11),ER(11),VALR(11),
1                  PKRB(11,11)NTCR(11),QR(11,11),ALLRB(11,11),
2                  NTRJ,WGTR(11),WR(11)

      COMMON/BLUE/NB(11),ALTCBR(11,11),EB(11),VALB(11),
1                  PKBR(11,11),NTCB(11),QB(11,11),ALLBR(11,11),
2                  NTBI,WGTB(11),WB(11),I1
```

With the exception of I1 in COMMON/BLUE/ these two commons are identical in definition for Red and Blue forces. The Blue definitions will be given here

NB	Number of weapons by type.
ALTCBR	Typical allocation of Blue weapons against Red.
EB	Engagement rates for each Blue weapon system.
VALB	Interface variable for other APP calculations.
PKBR	Probability that a Blue system kills a Red system.
NTCB	Typical number of Blue weapons assigned.
QB	Kill rate matrix of Blue against Red.
ALLBR	Allocation matrix of Blue against Red.
NTBI	Number of types of weapons, set to 11.
I1	Index weapon system.
WGTB	Interface variables for other APP calculations.
WB	Weighting factors for force ratio calculations.

f. COMMON/ENGMT/ERP(3,11),EBP(3,11),POSR,POSB,LCMBT, FRBCAS

ERP	Engagement rate for Red.
EBP	Engagement rate for Blue.
POSR	Combat posture for Red.
POSB	Combat posture for Blue.
LCMBT	Unit type identification of combat units.
FRBCAS	Force ratio Red to Blue limit for support requests.

g. COMMON/NAMES/NAMER(11),NAMEB(11)

NAMER	Names of Red combat systems.
NAMEB	Names of Blue combat systems.

h. COMMON/PRTFLG/PFLAG(20),PMOD(3),PDEBUG

PFLAG	Array of user control flags, see Figure III-4.
PMOD(1)	Message printout for this node only, if zero do all nodes.
PMOD(2)	Output start time.
PMOD(3)	Output stop time.
PDEBUG	Debug print flag, if one debugger is on.

i. COMMON/RAND/PRAND,PCDIS

PRAND	Pointer to queue of random distribution data sets.
PCDIS	Pointer to queue of random distribution indicators by message type.

j. COMMON/RANOUT/MSGONE,MSGCHG,MSGRPT,MSGPRT, LOCAS,NOCAS

MSGONE	Number of messages lost by communications.	(Send)
MSCHG	Number of message content changes made.	(MSGBAD)
MSGRPT	Number of requests to resend messages.	(MSGBAD)
MSGPRT	Number of messages too garbled to read.	(MSGBAD)
LOCAS	Number of CAS missions at wrong target.	(AIROPS)
NOCAS	Number of CAS missions at CAP.	(AIROPS)

k. COMMON/RULE/IRULE(6,400),RULE(3,400),MSGOUT(10,400), OUT(3,400)MSGIN(5,400),MIN(3,400),MSGOUT2(5,400), MSGQ(400,2),NORULE,NOMIN,NOMOUT

IRULE	Process variables; rule number, originator node type, time to do process, minimum messages required, schedule frequency, start time.
RULE	Process comment.
MSGOUT	First 10 output message parameters; rule number, message number, destination node type, priority, three link type and capacity pairs.

MSGOUT2	Last five output parameters; commander only flag, print output flag, two alternate node types, maximum age on the network.
OUT	Output message comment.
MSGIN	Input message parameters; rule number, message number, sending node type, maximum age message is useful, multiple use flag.
MIN	Input message comment.
MSGQ	Temporary work area to build rules; rule number, pointer to generic message output queue.
NORULE	Number of processes.
NOMIN	Number of rule input messages.
NOMOUT	Number of rule output messages.

2. Dynamic Data Structures

There are three sets of dynamic data structures (DDS) in C3EVAL. They are the NODE, CRC and GENERIC sets. A set has its root in a non-dynamic location and is linked together by pointers. As shown in Figures III-12, III-13 and III-14, the root locations are variables NODE, NCRC and NREDTE in COMMON/C3/. These structures are created during input by calls to Subroutine GIMME which acquires data blocks from dynamic memory. The length of each type data block is a fixed number in the code. This section lists the data blocks, defines their elements, gives the type of each variable, identifies the routine that creates the block and the ones that delete the block if applicable and specifies the location of the root. The symbols used in the DDSs documented in this section are:

Symbol	Type of Element
P	Pointer to another DDS
I	Integer value
R	Real value
C	Character

The DDS is traversed by the links shown in Figures III-12, III-13 and III-14. While each node is of a fixed length, the number of DDSs in a single queue is unlimited. Therefore, one node may have only one destination while another may have several.

When a specific data element is desired, the program code uses successive pointers to locate that element

Example: Find the capacity of communication link type 16 that connects node three to four.

1) Find node three by starting at COMMON/C3/NODE1. This is done in code by setting PNODE=NODE1. This is a location in DM. From the NODE DDS below, we see that location PNODE contains a pointer to the next NODE structure and that location PNODE+1 contains the unit identifier. This identifier is compared to the node number desired (three). If this is not the desired NODE, then PNODE is reset to MEMORY(PNODE), which is the location of the next node structure. This series is repeated until the desired NODE is found. (This can be done by Subroutine FIND.)

2) From the NODE DDS we see that PNODE+3 is a pointer to a communications destination queue. (The offset PNODE+3 is always one less than the number in the tables, because PNODE is the location of the first element in the DDS.) The DEST (destination) structure is traversed to find unit identifier four in the same manner that was used in step one.

3) The DEST DDS contains a pointer to its communications links. These links are traversed until a LINK DDS with type 16 is found.

4) The capacity of this link is located at PLINK+2.

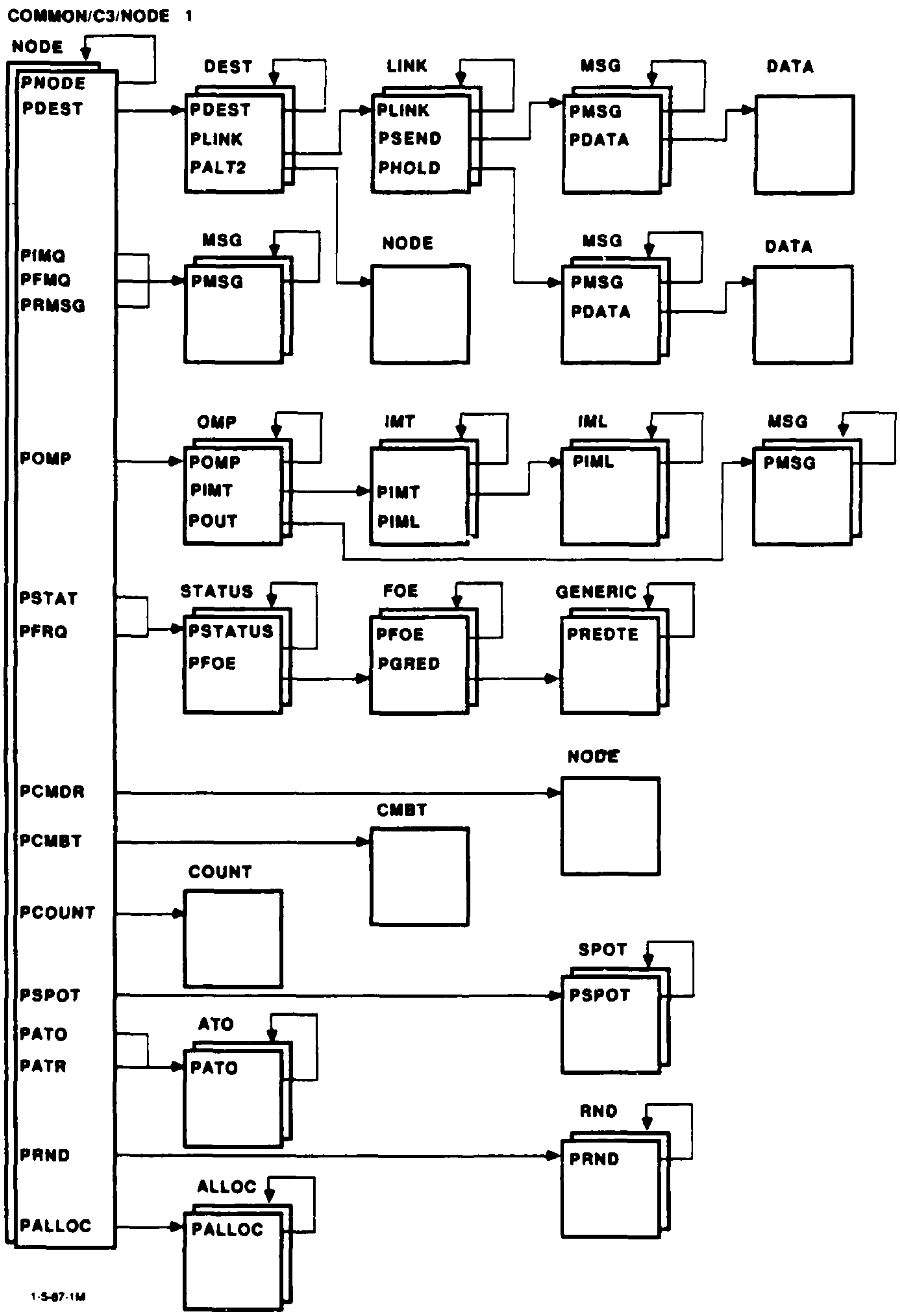


Figure III-12. NODE DYNAMIC LINKING

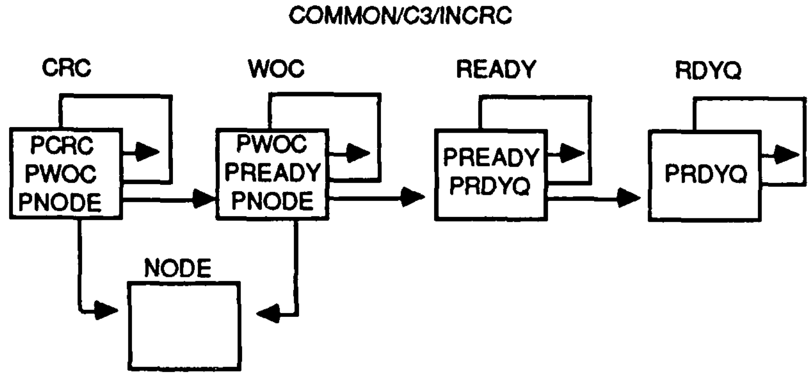


Figure III-13. DYNAMIC DATA LINKING

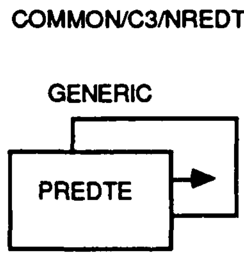


Figure III-14. GENERIC RED UNIT TABLES OF EQUIPMENT DYNAMIC DATA LINKING

Block Name: ALLOC   Block Size: 7

Use: This structure contains the allocation parameters for a commanding unit to use in approving requests for CAS by its subordinates.

Created by: INPUTA

Deleted by: N/A

Root: NODE(21)

Date: 31 Dec 86

Index	Element name	Type	Element meaning/use
1	PALLOC	P	Pointer to next allocation structure
2	TYPE	I	Type of allocation data (3000,3400)
3	SAT	I	Start allocation time
4	NAT	I	Number of allocation time periods
5	NSOR	I	Number of sorties allocated
6	MINSOR	I	Minimum sorties on a mission
7	MAXSOR	I	Maximum sorties to be allocated

Block Name: ATO Block Size: 21

Use: Contains the data portion of a request for CAS or the response to a CAS request.

Created by: MSGOUT

Deleted by: SEND

Root: MSG(18)

Date: 31 Dec 86

Index	Element name	Type	Element meaning/use
1	PATIO	P	Next air task order
2	ID	I	Support unit number
3	ETIME	I	Earliest support time or takeoff time
4	LTIME	I	Latest support time or on target time
5	ACTYPE	I	Type of aircraft
6	NUMAC	I	Number of aircraft or force ratio
7	RNO	I	Current process rule number
8-15	ID	I	Sequence of node IDs in network path to current node
16-21			Unused

Block Name: CMBT | Block Size: 32

Use: The combat data structure contains the combat weapon systems data for each combat level unit.

Created by: INPUTC

Deleted by: N/A

Root: NODE (9)

Date: 31 Dec 86

Index	Element name	Type	Element meaning/use
1	CFLAG	I	=0 no combat, =n > 0 n is period of combat
2	ENGB	I	Engagement rate of Blue: =0 not in combat, =1 high, =2 medium, =3 low intensity
3	ENGR	I	Engagement rate of Red: =0 not in combat, =1 high, =2 medium, =3 low intensity
4	FRB	R	Force ratio Red/Blue
5	POSB	I	Blue unit posture
6	POSR	I	Red unit posture
7-19	NB	R	Number of Blue weapons
20-32	NR	R	Number of Red weapons

Block Name: COUNT Block Size: 30

Use: This structure contains the message counters, limits and other parameters held for output at each node.

Created by: INPUT

Deleted by: N/A

Root: NODE(12) Date: 31 Dec 86

Index	Element name	Type	Element meaning/use
1		I	Not used
2	CIN	I	Number messages arrive on network
3	COUT	I	Number messages sent out on network
4	CHOLD	I	Number messages held by comm. limit
5	CKILL	I	Number messages killed by comm.limit
6	LINH	I	Input limit to hold messages
7	LINK	I	Input limit to kill messages
8	IIN	I	Number messages arrive through limit
9	IHOLD	I	Number messages held by input limit
10	IKILL	I	Number messages killed by input limit
11	LOUH	I	Output limit to hold messages
12	LOUK	I	Output limit to kill messages
13	OIN	I	Number messages sent out through limit
14	OHOLD	I	Number messages held by output limit
15	OKILL	I	Number messages killed by output limit
16			Unused
17	CAS	I	Number CAS sorties in combat
18			Unused
19	HELO	I	Number helicopter sorties in combat
20-30			Unused

Block Name:	CRC	Block Size:	5
Use:	Contains the parameters for a combat reporting center and the relationship to its noda structure and wing operations support unit.		
Created by:	INPUT		
Deleted by:	N/A		
Root:	/C3/NCRC	Date:	31 Dec 86

Index	Element name	Type	Element meaning/use
1	PCRC	P	Next CRC
2	ID	I	Unit number of CRC
3	TYPE	I	Unit type of CRC
4	PNODE	P	Pointer, CRC's node structure
5	PWOC	P	Pointer, wing operations structure

Block Name:	CUML	Block Size:	30
Use:	Structure contains cumulative sum of losses of Blue unit and cumulative losses of its foes.		
Created by:	COMBAT		
Deleted by:	NA		
Root:	/C3/LOSSUM	Date:	31 Dec 86

Index	Element name	Type	Element meaning/use
1	PCUML	P	Pointer to next CUML structure
2	NODE	I	Node number
3	ITIME	I	Combat time of cumulative of data
4	FRB	R	Force ratio
5-17	LOSSESB	R	Cumulative losses of Blue by weapon system
18-30	LOSSESR	R	Cumulative losses of Blue's foe by weapon system

Block Name:	DEST	Block Size:	9
Use:	This structure contains all of the other nodes which can be communicated with by the node to which it is attached.		
Created by:	INPUT		
Deleted by:	N/A		
Root:	NODE(4)	Date:	31 Dec 86

Index	Element name	Type	Element meaning/use
1	PDEST	P	Pointer to next destination element
2	ID	I	Destination unit identifier
3	PLINK	P	Pointer to communications queue
4	PALT1	P	Pointer to first alternate
5	PALT2	I	Pointer to second alternate
6	TYPE	I	Destination unit type
7	ALOFLG	I	Allocation flag, =1 allocated
8			Internal flag
9	SUBFLG	I	Subordinate flag, =1 subordinate

Block Name:	FOE	Block Size:	7
Use:	Identifies specific Red unit as foe of a subordinate and points to the generic Red unit table of equipment.		
Created by:	STATIN,INTLUP		
Deleted by:	INTLUP		
Root:	STATUS(27)	Date:	31 Dec 86

Index	Element name	Type	Element meaning/use
1	PREDTE	P	Pointer to next FOE structure
2	UTYPE	I	Type of Red unit
3-5	UNITID	C	Red unit name
6	PGRED	P	Pointer to generic FOE
7	TIME	I	Time of unit identification

Block Name: GENERIC

Block Size: 17

Use: Structure for table of equipment for a generic Red unit.

Created by: INPUTC

Deleted by: N/A

Root: /C3/NREDTE

Date: 31 Dec 86

Index	Element name	Type	Element meaning/use
1	PREDTE	P	Pointer to next generic Red TE
2	UTYPE	I	Type of Red unit
3-15	SYSTEM	R	Number of combat weapon systems
16	ENGRAT	I	Engagement rate
17	POSTUR	I	Combat posture

Block Name:	HOC		Block Size: 11
Use:	Structure contains data on helicopters on the ground and parameters that affect their assignment to support missions.		
Created by:	INPUTA		
Deleted by:	NA		
Root:	SUP (2)		Date: 31 Dec 86

Index	Element name	Type	Element meaning/use
1	PHOC	P	Pointer to next HOC
2	NSOR	I	Number helicopters normally assigned
3	PASSGN	P	Pointer to ATO assigned helicopter support
4			Not used
5	ATIME	I	Alert response time, not used
6	ETIME	I	Enroute time
7	MINH	I	Minimum helicopters on mission
8	PS	R	Probability of survivability enroute
9	PREADY	P	Helicopter ready queue
10	RTIME	I	Ground turnaround time, =1
11	NDIST	I	Random distribution for helicopter unit

Block Name:	IML	Block Size:	5
Use:	The input message list contains the variables that indicate specific message receipt from a designated unit (NODE).		
Created by:	MSGIN		
Deleted by:	N/A		
Root:	IMT(4)	Date:	31 Dec 86

Index	Element name	Type	Element meaning/use
1	PIML	P	Next input message list
2	NODE	I	Unit identifier of originator
3	MFLAG	I	Flag, message was received
4	MAGE	I	Age of last input message
5	OTIME	I	Time last input was originated

Block Name: IMT Block Size: 6

Use: Input message type structure contains the parameters required to process a specific input message type that originated at a specific unit type. Each IMT has an IML queue.

Created by: RULEIN

Deleted by: N/A

Root: OMP(3)

Date: 31 Dec 86

Index	Element name	Type	Element meaning/use
1	PIMT	P	Next input message type
2	MSG	I	Type of input message
3	OTYPE	I	Type unit that originates message
4	PIML	P	Input message list
5	MAXAGE	I	Maximum age of useful message
6	USE	I	Message use if 0 only once

Block Name: INTEL REPORT

Use: *Structure for foe identification message type 3136.*

Block Size: 21

Created by: INTLUP

Deleted by: ATODEL,INTLUP

Root: MSG(18)

Date: 31 Dec 86

Index	Element name	Type	Element meaning/use
1	PDATA	P	Pointer to next data report
2	ORIG	I	Node identifier of Blue combat unit
3	TIME	I	Time report data was created
4	TYPE	I	Type of foe unit
5	POSTUR	I	Posture of foe unit
6-8	NAME	C	Name of specific unit
9	PINTEL	P	Pointer to next INTEL report
10-16			Unused
17	PROCES	I	Input process number
18-21			Unused

Block Name:	LINK		Block Size: 7
Use:	This structure contains the parameters for a link between the node and the destination in the root DEST structure.		
Created by:	INPUT		
Deleted by:	N/A		
Root:	DEST(3)		Date: 31 Dec 86

Index	Element name	Type	Element meaning/use
1	PLINK	P	Pointer to next link element
2	LTYPE	I	Type of link
3	LCAP	I	Capacity of link
4	PSEND	P	Pointer to message send queue
5	PHOLD	P	Pointer to message hold queue
6	PWORK	P	Temporary queue for allocation
7	NDIST	I	Distribution type for lost message

Block Name: MSG Block Size: 21
Use: Basic message structure for all messages.

Created by: ATORTN,EXTMSG,MAXMSG,MSGOUT,RULEIN
Deleted by: MDELAY,MINLIM,MSGIN,NODE,SEND
Root: NODE(6),(7),(23);LINK(4),(5);IMT(4),OMP(4) Date: 31 Dec 86

Index	Element name	Type	Element meaning/use
1	PMSG	P	Pointer to next message element
2	STIME	I	Time to send messages
3	MTYPE	I	Type of message
4	GTYPE	I	Originator of message
5	DEST1	I	Destination unit identifier
6	PRIOR	I	Priority of message
7	LTYPE1	I	Primary path type
8	CAP1	I	Capacity required on path 1
9	LTYPE2	I	First alternate path type
10	CAP2	I	Capacity required on path 2
11	LTYPE3	I	Last alternate type
12	CAP3	I	Capacity required on path 3
13	DEST2	I	Alternate unit type
14	CMDR	I	=1 destination is commander only
15	FLAG1	I	Output flag
16	FLAG2	I	Alternate transmission flag
17	DEST3	I	Alternate unit type
18	PDATA	P	Pointer to additional data
19	NODE	I	Unit identifier of originator
20	OTIME	I	Time message was created
21	MAXAGE	I	Maximum time to be on network

Block Name: NODE Block Size: 25

Use: This structure is the top element for each node in the game. It is used to locate the functional structures that contain the node's characteristics.

Created by: INPUT

Deleted by: N/A

Root: /C3/NODE1

Date: 31 Dec 86

Index	Element name	Type	Element meaning/use
1	PNODE	P	Pointer to next node
2	ID	I	Unit identifier number
3	TYPE	I	Unit type
4	PDEST	P	Pointer to destination queue
5	POMP	P	Pointer to output message process
6	PIMQ	P	Pointer to input message queue
7	PFMQ	P	Pointer to future message queue
8	PCMDR	P	Pointer to commander
9	PCMBT	P	Pointer to combat data structure
10	PAOPS	P	Pointer to assigned air support
11	PATO	P	Pointer to air requests
12	PCOUNT	P	Pointer to counters for output
13-15	NAME	C	Unit name
16	PATR	P	Pointer to acknowledged requests
17	PFSO	P	Pointer to requested fire support
18	PFSR	P	Pointer to approved fire support
19	PSTAT	P	Pointer to subordinate status queue
20	PSPOT	P	Pointer to spot report queue
21	PALLOC	P	Pointer to allocation parameters
22	PFRQ	P	Pointer to force ratio queue
23	PRMQ	P	Pointer to random message queue
24	PRND	P	Pointer to random distribution indicators
25	PSUP	P	Pointer to support data structure

Block Name: OMP Block Size: 11

Use: The output message process structure contains the parameters for message handling. Each OMP has an IMT queue and an MSG QUEUE.

Created by: RULEIN

Deleted by: N/A

Root: NODE(5)

Date: 31 Dec 86

Index	Element name	Type	Element meaning/use
1	POMP	P	Next output message type
2	RNO	I	Rule number
3	PIMT	P	Input message type queue
4	POUT	P	Output message queue
5	LTIME	I	Last time this process complete
6	OTYPE	I	Type unit that originated process
7	CTIME	I	Time to complete process
8	MFLAG	I	>0 minimum current message to do this process
9	TFLAG	I	=0 not periodic >0 periodic interval
10	PRULES	P	Data element for action
11	STIME	I	Start time of periodic process

Block Name: RAND — Block Size: 13

Use: This structure contains a random distribution of a specific type for use by Subroutine RANDOM.

Created by: RANDIN

Deleted by: N/A

Root: /RAND/PRAND

Date: 31 Dec 86

Index	Element name	Type	Element meaning/use
1	PNEXT	P	Pointer to next RAND block
2	NDIST	I	Distribution type
3-13	Value	R	Distribution values at 0, .1, .2 , .3, .4, .5, .6, .7, .8, .9, 1.0

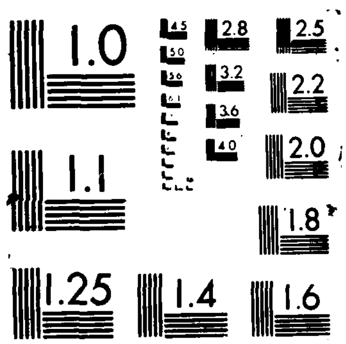
Block Name:	RDI	Block Size:	4
Use:	Random distribution indicator structure contains the message type and the indicators for partial message and data corruption random distributions.		
Created by:	RANDIN		
Deleted by:	N/A		
Root:	/RAND/PCDIS	Date:	31 Dec 86

Index	Element name	Type	Element meaning/use
1	PNEXT	P	Pointer to next RDI structure
2	MSGNUM	I	Message type
3	MPDIS	I	Indicator of partial message distribution
4	MCDIS	I	Indicator of message content distribution

Block Name:	RDYQ		Block Size: 3
Use:	The aircraft availability que contains the number of a specific aircraft type that will be available for takeoff at a specified time.		
Created by:	AIROPS,INPUTA		
Deleted by:	N/A		
Root:	READY(3)		Date: 31 Dec 86

Index	Element name	Type	Element meaning/use
1	PRDYQ	P	Next aircraft element
2	ACTIME	I	Time aircraft will be ready
3	NUMAC	R	Number of aircraft to be ready

AD-A186 370 C3EVAL MODEL DEVELOPMENT--1986 VOLUME 2 PROGRAMMERS' 3/4
MANUAL(U) INSTITUTE FOR DEFENSE ANALYSES ALEXANDRIA VA
R F ROBINSON ET AL. APR 87 IDA-P-1978-VOL-2
UNCLASSIFIED IDA/HQ-86-31573 MDA903-84-C-0031 F/G 12/5 NL


1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

Block Name: READY		Block Size: 3
Use:	Identifies a specific aircraft type under control of the WOC.	
Created by:	INPUTA	
Deleted by:	N/A	
Root:	WOC(9)	Date: 31 Dec 86

Index	Element name	Type	Element meaning/use
1	PREADY	P	Next aircraft element
2	ACTYPE	I	Aircraft type
3	PRDYQ	P	Aircraft ready queue

Block Name: RND Block Size: 4

Use: This random distribution indicator structure contains the indicators for message distributions.

Created by: INPUT

Deleted by: N/A

Root: NODE(24) Date: 31 Dec 86

Index	Element name	Type	Element meaning/use
1	-		Not used
2	MLEN	I	Indicator of message length distribution
3	MDEL	I	Indicator of message delay distribution
4	CASD	I	Indicator of CAS request delay distribution

Block Name: STATUS

Use: Status of subordinate units, created if subordinate flag in input =2.

Block Size: 43

Created by: INPUT

Deleted by: N/A

Root: NODE(19)

Date: 31 Dec 86

Index	Element name	Type	Element meaning/use
1	PSTAT	P	Pointer to next status structure
2	UNITID	I	Identifier of status unit
3	TIMEB	I	Last time befor . tatus updated
4-14	NB	I	Number of Blue weapons
15-25	BLOSS	R	Number of Blue losses
26	TIMER	I	Last time Red status updated
27	PFOE	P	Pointer to Red foe structure
28-38	RLOSS	R	Number of Red losses
39	POSB	I	Blue posture
40	POSR	I	Red posture
41	FRB	R	Red to Blue force ratio
42	PFRQ	P	Pointer to next status structure in force ratio queue
43	LFRT	I	Last time force ratio calculated

Block Name:	SPOT REPORT	Block Size:	21
Use:	Structure for combat loss date for message types 3126 and 3130.		
Created by:	RPTLOS		
Deleted by:	ATODEL		
Root:	MSG(18) and NODE(20)	Date:	31 Dec 86

Index	Element name	Type	Element meaning/use
1	PSPOT	P	Pointer to next data report
2	ORIG	I	Node identifier to report originator
3	TIME	I	Time report was created
4	FOE	I	Flag: =0 foe, -1 subordinate data
5-15	LOSS	R	Losses of combat systems
16	POSTUR	I	Posture of unit
17	PROCES	I	Input process number
18-21			Unused

Block Name:	SUP		Block Size: 4
Use:	This structure is the root for a node's helicopter and artillery combat support queues.		
Created by:	INPUTA		
Deleted by:	NIA		
Root:	NODE(25)		Date: 31 Dec 86

Index	Element name	Type	Element meaning/use
1	PSUP	P	Pointer to next SUP, not used
2	PHOC	P	Pointer to HOC structure
3	PAOC	P	Pointer to AOC structure
4	PATOQ	P	Pointer to local node queue of requests for support (3400)

Block Name:	WOC
Use:	Structure contains data on aircraft on the ground and parameters that effect their assignment to CAS missions.
Created by:	INPUTA
Deleted by:	N/A
Root:	CRC(5)

Block Size: 11

Date: 31 Dec 86

Index	Element name	Type	Element meaning/use
1	PWOC	P	Next WOC
2	ID	I	Unit number of WOC
3	TYPE	I	Unit type ofWOC
4	PNODE	P	WOC's node structure
5	ATIME	I	Alert response time
6	ETIME	I	Enroute time
7	MINAC	I	Minimum aircraft on mission
8	PS	R	Probability of survivability enroute
9	PREADY	P	Aircraft ready queue
10	RTIME	I	Ground turn-around time
11	NDIST	I	Random distribution type for CAS support

G. PROGRAM NOTES

This section contains notes on unique situations in the C3EVAL model and its data bases.

During the development of the code it became convenient to preassign certain data values. These numbers may be required in the data sets and their use must coincide with the definitions below:

Node number for CAP	99
Node type for WOC	7000
Message numbers for external air requests	3000,2900
Message number for force ratio support requests	3400
Message number for CAS notifications	7000
Message number for Red losses	3126
Message number for Blue losses	3130
Message number for foe's identification	3136
Process numbers for random created messages	3800,4800,5800,6800, 7800,5900,7900
Internal message number for request resent message	9991
Internal message number to delete ATO structures	9990,9993

Routine RULPRT has the types of nodes in a data statement. If additional types are used, they will not be printed out on File C3ECHO unless this array is changed.

The following comments pertain to input values. File C3DATA has a data set "LMNO." The first value is the input threshold to hold messages that exceed this count. However, all messages input at the node that have the same priority are treated the same. For example, if the hold limit is five and there are three priority 1 messages, three priority two messages and three priority three messages, the algorithm will allow all three priority one's and all three priority two's to be input because the limit fell in the priority two count range. All messages with priorities higher than two will be held (if their data useful time is not exceeded) or deleted. The priority level algorithm is also applied to messages at the delete limit. The limits are cumulative (i.e., if the limits are five and nine, the number of candidates to be held is four).

Post processing graphics read Files TIMET and LOSST to get their input. These files are created by C3EVAL if the graphic flag is on in C3DATA, Flag Number 14. Red and Blue weapon strengths and combat postures may be changed during the game. File CBDATA would have two additional lines of data (one for Red and one for Blue) with the desired times for the change. All 11 systems are modified by this input (a minus sign removes forces), with the exception of Blue CAS. This field is not used. All CAS for Blue must be requested through the network and is subject to aircraft availability of the WOC.

Message types that are created by successful completion of a decision rule are used to create specific messages at a node and are placed on the message type's primary destination path and communications type to the receiving node. If the primary path and communications type does not exist (note capacity may be zero but the path exists), then the message will be deleted and an error written on C3TIME.

H. INTERNAL CODE DOCUMENTATION

```
SUBROUTINE AIROPS
****************************************************
* REPRESENTS THE WOC AND CRC ACTIONS
* INCLUDES RANDOM TARGET ASSIGNMENT CAPABILITY
****************************************************
INCLUDE COMMONS
*****************************************************
MAKE READY QUEUES CURRENT
DO FOR ALL CRC'S
   DO FOR ALL WOC'S
      REDUCE READY QUEUE
       GET NEXT A/C READY QUEUE
    *************************************************
    SCHEDULE MISSION TAKE OFFS
    CHECK AIR SUPPORT REQUESTS
       CHECK FOR LAST USABLE TIME FOR ATO
          FIND A/C TYPE
          CHECK ENOUGH AIRCRAFT AVAILABLE
             CALCULATE NUMBER ASSIGNED
         TEST FOR RANDOM UNIT TO SUPPORT
               SORTIES SENT TO CAP
                     SORTIES SENT TO DIFFERENT TARGET
                     CORRECT CAP COUNTER
             PUT ATO ON CRC'S LIST
             MAKE MESSAGE FOR CORPS
             PUT MESSAGE ON FUTURE QUEUE
             WRITE ACTION ON OUTPUT
          AIR REQUEST WILL NOT BE FILLED
          SET NUMBER OF AIRCRAFT TO ZERO
          REMOVE FROM WOC'S ATO LIST
          SEND MESSAGE TO CORPS
       GET NEXT ATO
       ATO USED CASE
       ATO NOT USED CASE
    END OF THIS WOC SCHEDULE ACTION, PRINT STATUS
    GET NEXT WOC
 LAST WOC FOR THIS CRC COMPLETED SCHEDULE
 SET CAS FOR THIS TIME CYCLE
 DO FOR ALL NODES
    CHECK FOR COMBAT TYPE UNIT
       SET CAS SORTIES TO ZERO
    GET NEXT NODE
 CHECK FOR WOC SOURCE
 CHECK TIME ON TARGET
    SET CAS SUPPORT
    INCREMENT CAS SORTIE COUNT
    RESET UNIT AIR REQUEST TO NULL
    SCHEDULE SORTIE LANDING
    FIND WOC
    GET NEXT ATO
 LAST CAS SUPPORT PUT IN COMBAT, GET NEXT CRC
LAST CRC SCHEDULED
```

```
SUBROUTINE ALLOC(ALBCX,NBCY,NY,NTX,NTY,AX)
*************************************************************
*  THIS ROUTINE COMPUTES THE ACTUAL CASE ALLOCATION MATRICES
*************************************************************
ALLOCATION MATRICES ARE COMPUTED EXCLUSIVE OF METHOD SUBROUTINES
```

```
SUBROUTINE ALOCAS (PNODE,POMP,POUT,NALLOC,MINSOR,PATOQ)
*****************************************************
*  ALOCAS ALLOCATES AVAILABLE SORTIES TO REQUESTS FOR
*  CAS USING THE SUBORDINATES RED/BLUE FORCE RATIO QUEUE
*  TO SET PRIORITY
*****************************************************
*  INPUT
*     PNODE - POINTER TO NODE
*     POMP  - POINTER TO OUTPUT TYPE
*     POUT  - POINTER TO MESSAGE TYPE
*     NALLOC- NUMBER OF SORTIES TO ALLOCATE
*     MINSOR- MINIMUM OF SORTIES FOR A MISSION
*     PATOQ - POINTER TO TEMPORARY ATO ALLOCATION QUEUE
*****************************************************
DO WHILE SORTIES ARE AVAILABLE
   DO FOR EACH SUBORDINATE
         CHECK NUMBER OF SORTIES
         ADD SORTIES TO STATUS
      GET NEXT SUBORDINATE
   SEND REQUEST DENIED TO SUBORDINATES
         CREATE 7000 MESSAGE WITH ZERO SORTIES
      GET NEXT FRQ
   RESET NUMBER ALLOCATED TO ACTUAL NUMBER
   DELETE ALL ENTRIES IN TEMPORARY ATO QUEUE
```

```
SUBROUTINE ALOCAT
*****************************************************
*  ALLOCATE OUTPUT TO LINKS USING CAPACITY AND DIRECT
*  COMMUNICATIONS ONLY, AND KILL OLD MESSAGES ON THE
*  FUTURE MESSAGE QUEUE
*****************************************************
*  CALLS - FIND, ERROUT, SNAP, RELEAS
*****************************************************
INCLUDE COMMONS
GET FIRST NODE
DO FOR ALL NODES
   DO FOR EACH MESSAGE ON FUTURE QUEUE
      CHECK TIME
         CHECK AGE OF MESSAGE
            DELETE THIS MESSAGE
               DELETE DATA STRUCTURE
            DELETE MESSAGE STRUCTURE
            GET NEXT MESSAGE
         MOVE MESSAGE TO HOLD QUEUE
            ERROR IN ROUTING
         DESTINATION FOUND
         FIND LINK TYPE
            ERROR IN ROUTING
   GET NEXT MESSAGE
   DO FOR ALL DESTINATIONS
      DO FOR ALL LINKS
         FIND LAST SEND MESSAGE
         LAST MESSAGE FOUND
         MOVE HOLD QUEUE
         LOCATE LAST SEND MESSAGE IN CAPACITY
            LAST MESSAGE FOUND
            MOVE REMAINING MESSAGES TO HOLD QUEUE
         END OF SEND QUEUE CORRECTION
      GET NEXT LINK
   GET NEXT DESTINATION
   END OF SEND QUEUE JUSTIFICATION
GET NEXT NODE
LAST NODE COMPLETED
```

```
SUBROUTINE ALTOUT
****************************************************
*  REVIEW MESSAGES IN HOLD QUEUES TO SEE IF THEY SHOULD
*  BE SENT VIA ALTERNATIVE DESTINATIONS
****************************************************
*  CALLS MOVMSG, SNAP, FIND
****************************************************
INCLUDE COMMONS
DO FOR ALL NODES
   DO FOR ALL DESTINATIONS
      SET ALLOCATION FLAG OFF
      CHECK NODES ALTERNATES TO THIS DESTINATION
         DO FOR ALL LINKS TO THIS DESTINATION
            DO FOR ALL MESSAGES ON THIS HOLD QUEUE
               MOVE MESSAGES FROM PHOLD TO ALTERNATE1
               BY FIRST MESSAGE ALTERNATE
               RECHECK HOLD QUEUE
               CHECK IF ALTERNATE 2 EXISTS
               MOVE MESSAGES FROM PHOLD TO ALTERNATE 2
               BY FIRST MESSAGE ALTERNATE
               RECHECK HOLD QUEUE
               MOVE MESSAGES FROM PHOLD TO ALTERNATE 1
               BY SECOND MESSAGE ALTERNATE
               CHECK IF ALTERNATE 2 EXISTS
               RECHECK HOLD QUEUE
               MOVE MESSAGES FROM PHOLD TO ALTERNATE 2
               BY SECOND MESSAGE ALTERNATE
            GET NEXT LINK

     GET NEXT DESTINATION
  GET NEXT NODE
LAST NODE
```

```
SUBROUTINE ATOALO (PNODE, POMP,POUT)
****************************************************
*  ATOALO ALLOCATES SYSTEMS TO SUBORDINATES TO
*  SUPPORT REQUESTS, MESSAGE TYPES 2900, 3000, 3400
****************************************************
*  INPUT
*     PNODE - POINTER TO NODE
*     POMP  - POINTER TO OUTPUT TYPE
*     POUT  - POINTER TO MESSAGE TYPE
****************************************************
INCLUDE COMMONS
GET FIRST ATO FOR THIS NODE
INITIALIZE POINTER FOR INTERNAL ATO QUEUE
DO FOR EACH ATO MESSAGE
   IF PROCESS MATCHES ATO, CREATE MESSAGE
      FIND EXISTING ENTRY IN INTERNAL ATO QUEUE
         SUPPORT NODE NOT ON QUEUE, CREATE ENTRY
      CONSOLIDATE SUPPORT DATA IN EXISTING ENTRY
   GET NEXT ATO
CHECK ATOQ
   CALCULATE PERCEIVED FORCE RATIOS
   SORT SUBORDINATE STATUS STRUCTURES FORCE RATIO QUEUE
      NODE DOES NOT ALLOCATE THIS TYPE, PROCESS ALL ATO'S
      TEST FOR CORRECT SET OF ALLOCATION PARAMETERS
         NOT CORRECT SET OF ALLOCATION PARAMETERS, GET NEXT SE
   ALLOCATION PARAMETERS FOUND
   TEST FOR CURRENT ALLOCATION TIME PERIOD
   CALCULATE NUMBER OF SORTIES TO BE ALLOCATED
   ALLOCATE SORTIES AVAILABLE
   LOG ACTUAL NUMBER OF SORTIES ALLOCATED
```

```
SUBROUTINE ATODEL (PNODE, POMP)
*******************************************************
*  ATODEL DELETES ALL ATO REQUEST AND REPLY BLOCKS THAT
*  HAVE ITS PROCESS NUMBER
*******************************************************
*  CALLS - GIMME, SNAP, FIND, ERROUT, RULES
*******************************************************
*  INPUT
*     PNODE - POINTER TO NODE
*     POMP  - POINTER TO OUTPUT TYPE
*******************************************************
INCLUDE COMMONS
SKIP ALL DELETIONS FOR THE WOC
DELETE REQUESTS

   TEST FOR THIS PROCESS NUMBER
DELETE REPLYS
   TEST FOR THIS PROCESS NUMBER
DELETE SPOT REPORTS
   TEST FOR THIS PROCESS NUMBER
```

```
SUBROUTINE ATOLIST
 INCLUDE COMMONS
```

```
SUBROUTINE ATOOUT (PNODE, POMP,POUT)
*******************************************************
*  ATOOUT CREATES OUTPUT MESSAGES THAT PASS ALONG AIR
*  SUPPORT REQUESTS, MESSAGE TYPES 2900, 3000, 3400
*******************************************************
*  CALLS - GIMME, SNAP, FIND, ERROUT, RULES
*******************************************************
*  INPUT
*     PNODE - POINTER TO NODE
*     POMP  - POINTER TO OUTPUT TYPE
*     POUT  - POINTER TO MESSAGE TYPE
*******************************************************
INCLUDE COMMONS
GET FIRST ATO FOR THIS NODE
   IF PROCESS MATCHES ATO, CREATE MESSAGE
         ADD SORTIES TO STATUS
```

```
SUBROUTINE ATORTN (PNODE, POMP,POUT)
******************************************************
*  ATORTN PASSES ALONG AIR SUPPORT APPROVALS TO THE
*  REQUESTOR USING MESSAGE TYPE 7000
******************************************************
*  CALLS - GIMME, SNAP, FIND, ERROUT, RULES
******************************************************
*  INPUT
*     PNODE - POINTER TO NODE
*     POMP  - POINTER TO OUTPUT TYPE
*     POUT  - POINTER TO MESSAGE TYPE
******************************************************
   INCLUDE COMMONS
   CHECK FOR ATO BLOCK
   CHECK ATO FOR THIS PROCESS
      GET AIR SUPPORT RETURN DESTINATION FROM ATO LIST
      FIND RETURN DESTINATION
   CREATE OUTPUT MESSAGE
   ENTER DATA
   SET ALTERNATE DESTINATIONS
      SET FIRST ALTERNATE TO FIRST NODE ALTERNATE
   TRY SECOND ALTERNATE
      SET FIRST ALTERNATE TO SECOND NODE ALTERNATE
         SET SECOND ALTERNATE TO NODES FIRST ALTERNATE
   TRY SECOND MESSAGE ALTERNATE
      SET FIRST ALTERNATE TO FIRST NODE ALTERNATE
   TRY SECOND NODE ALTERNATE
      SET FIRST ALTERNATE TO SECOND NODE ALTERNATE
   TRY SECOND NODE ALTERNATE
      SET SECOND ALTERNATE TO SECOND NODE ALTERNATE
   END OF ALTERNATE LOGIC
   CHECK FOR MESSAGE TRACKING FLAG
      SET TRACKING FLAG
   CHECK FOR MESSAGE SEND TIME
      PUT MESSAGE ON FUTRUE QUEUE BY TIME
   PUT MESSAGE ON SEND QUEUE BY PRIORITY
      GET SEND QUEUE
      FIND LINK TYPE
   GET NEXT ATO
   RESET MESSAGE TRACKING FLAG OFF
```

```
SUBROUTINE ATO_IN (PNODE, POMP)
******************************************************
*  ATO_IN TESTS A COMMANDER NODE TO DETERMINE IF IT
*    ALLOCATES RESOURCES IN RESPONSE TO MESSAGE TYPE 3400.
*  IF NOT ATOOUT IS CALLED TO PROCESS THE OUTPUT FOR THE RULE.
*  IF THE NODE DOES, ATO_IN CREATES A TEMPORARY QUEUE OF
*    SUPPORT REQUESTS IN NODE>SUP>ATOQ, CALCULATES THE
*    PERCIEVED FORCE RATIO AND SORTS THE STATUS QUEUE.
******************************************************
*  INPUT
*     PNODE - POINTER TO NODE
*     POMP  - POINTER TO OUTPUT TYPE
******************************************************
INCLUDE COMMONS
GET FIRST ATO FOR THIS NODE
INITIALIZE POINTER FOR INTERNAL ATO QUEUE
DO FOR EACH ATO MESSAGE
   IF PROCESS MATCHES ATO, COLLAPSE REQUESTS
IF INTERNAL QUEUE IS EMPTY, INITIALIZE THE QUEUE
      FIND EXISTING ENTRY IN INTERNAL ATO QUEUE
         SUPPORT NODE NOT ON QUEUE, CREATE ENTRY
      CONSOLIDATE SUPPORT DATA IN EXISTING ENTRY
  SELECT THE MAXIMUM FORCE RATIO REPORTED
   GET NEXT ATO
CHECK ATOQ
CALCULATE PERCEIVED FORCE RATIOS
SORT SUBORDINATE STATUS STRUCTURES FORCE RATIO QUEUE
PUT TEMPORARY REQUEST QUEUE IN NODE>SUP>ATOQ
```

```
BLOCK DATA
***************************************************
* DATA FOR THE GENERIC TYPE UNIT COMBAT DATA
* FOR BOTH RED AND BLUE FORCES
***************************************************
INCLUDE COMMONS
ENGAGEMENT RATES
UNIT POSTURES
TYPICAL NUMBER OF TYPES IN A UNIT
CONTROL SIZES
ALLOCATION OF RED AGAINST BLUE
ALLOCTATION OF BLUE AGAINST RED
PK RED AGAINST BLUE
PK BLUE AGAINST RED
```

```
 PROGRAM C3EVAL
*********************************************************
*    MAIN PROGRAM, TEST NODE     REVISED MAY 6, 1985
*    CALLS DMINIT, INPUT, EVENTS, OUTPUT, RESTOR
*********************************************************
INCLUDE COMMONS
*INITIALIZE I/O UNITS
*INITIALIZE DYNAMIC MEMORY
 RESTART RUN
*INPUT RANDOM DISTRIBUTIONS
*INPUT SCENARIO AND TIME ZERO DATA
 INPUT RULES AT TIME ZERO
*SIMULATE EVENTS
*PRODUCE FINAL REPORTS AND SAVE STATUS
*WRITE MESSAGE SUMMARY ON C3SUM
*WRITE VIRTUAL MEMORY USE ON C3SUM
*WRITE COMMANDER'S PERCEPTIONS ON C3SUM
*WRITE CUMULATIVE LOSSES ON C3SUM
```

```
SUBROUTINE CASLOS(SHOTB)
*****************************************************
*  CASLOS SUBTRACTS COMBAT LOSSES DUE TO CAS FROM
*  RETURNING AIRCRAFT
*****************************************************
INCLUDE COMMONS
   DO FOR ALL WOC'S
      SET TIME
      CHECK AIRCRAFT TYPE
            FIND CORRECT TIME
                  GET NEXT READY QUEUE
               REDUCE NUMBER OF AIRCRAFT RETURNING
            ERROR (NO SORTIES RETURNING)
         INCREMENT PREADY
         INCREMENT PWOC
      INCREMENT PCRC
   ERROR (NO TYPE 1 AIRCRAFT RETURNING)
```

```
SUBROUTINE CHFORC(NRC,NBC)
*****************************************************
*  INPUT COMBAT VALUES IN NRC AND NBC
*    1 - TIME FOR UNIT REINFORCEMENTS OR ENGAGEMENT
*        RATE CHANGE
*    2 - UNIT NUMBER
*    3 - ENGAGEMENT RATE OF UNIT,0 = RESERVE,
*                              ELSE = RATE INDEX
*    4 - UNIT POSTURE
*    5-17 - NUMBER OF COMBAT SYSTEM REINFORCEMENTS
*****************************************************
INCLUDE COMMONS
```

```
SUBROUTINE CHLIM(LIMIT)
**************************************************
* CHANGE INPUT OR OUTPUT MESSAGE LIMITS AT A NODE
**************************************************
INCLUDE COMMONS
FIND NODE
UPDATE RANDOM DISTRIBUTION TYPES
```

```
SUBROUTINE CHLINK(LINKT)
*****************************************************
* CHANGE LINK CAPACITIES DURING TIME T
*****************************************************
INCLUDE COMMONS
```

```
SUBROUTINE CHNODE(NODET)
**************************************************
* CHANGE COMMANDER OR SUBORDINATE NODE
**************************************************
INCLUDE COMMONS
```

```
SUBROUTINE COMBAT
*************************************************************
*  INTERFACE TO METHOD 4 CALCULATION OF APP AND THEN
*  CALCULATES THE COMBAT DRAW DOWN
*  REQUESTS SUPPORT IF COMBAT RATIO LOW AND WAIT TIME BETWEEN
*    REQUESTS HAS PAST, THIS REQUEST MAY HAVE A RANDOM DELAY
*************************************************************
INCLUDE COMMONS
DO FOR ALL NODES
      CALCULATE FORCE RATIO FOR COMBAT INTERACTIONS
  SET UNIT ENGAGEMENT RATES
  SET UNIT POSTURES
  CALCULATE COMBAT LOSSES
  CALCULATE REDS KILLED
  CALCULATE BLUES KILLED
      RECALCULATE FORCE RATIO FOR COMBAT REACTION
      CREATE SUPPORT REQUEST IF FORCE RATIO REQUIRES
         CHECK FOR ACTIVE ATO IN EXISTANCE
            REQUEST SUPPORT
            TEST FOR RANDOM DELAY INDICATED
            PUT MESSAGE ON FUTURE QUEUE IN TIME SEQUENCE
  SUBTRACT CAS LOSSES FROM RETURNING SORTIES
  SEND COMBAT LOSS REPORT
  SAVE LOSSES IN LOSSUM QUEUE
  PUT CURRENT STRENGTHS INTO UNIT'S CMBT STRUCTURE
```

```
      SUBROUTINE CONTRL
*********************************************************
* READ IN MODE OF OPERATION AND PRINT CONTROL FLAGS
*   PRINT FLAG (PFLAG(I)) DEFINITIONS
*      1    ALL MESSAGES AT ALTERNATE DESTINATION
*      2    ALL MESSAGES ON INPUT QUEUES
*      3    ALL MESSAGES ON OUTPUT QUEUES
*      4    ALL MESSAGES ON FUTURE QUEUES
*      5    ALL MESSAGES BEING HELD
*      6    ALL MESSAGES DELETED
*      7    STATUS OF RULE STRUCTURE
*      8    COMBAT SUPPORT SCHEDULED
*    9-10   NOT ASSIGNED
*     11    TRACKED MESSAGES AT ALTERNATE DESTINATIONS
*     12    TRACKED MESSAGES ON INPUT QUEUES
*     13    TRACKED MESSAGES ON OUTPUT QUEUES
*     14    TIME T OUTPUT ON FILE 14 REQUIRED
*     15    COMBAT LOSS VECTORS
*     16    FORCE RATIO CALCULATIONS
*     17    RULE STATUS AT FINAL TIME
*     18    AUTO POSTURE REQUIRED
*     19    RANDOM PROCESSING REQUIRED
*     20    USED INTERNALLY FOR SUM OF FLAGS
*******************************************************
*   PRINT MODIFIER (PMOD(I)) DEFINITIONS
*      1    OPTIONAL OUTPUT RESTRICTED TO THIS NODE
*      2    OPTIONAL OUTPUT STARTS AT THIS TIME
*      3    OPTIONAL OUTPUT STOPS AFTER THIS TIME
*********************************************************
      INCLUDE COMMONS
```

```
SUBROUTINE CS_ALO(NTYPE,PNODE,PA,PALLOC,IHELP)
*****************************************************
*  ALLOCATE COMBAT SUPPORT SYSTEMS
*    PNODE  - POINTER TO UNIT
*    INDXFR - INDEX ARRAY SORTED BY INCREASING FORCE RATIO FOR
*             COMBAT SUPPORT TYPE 1 = CAS, 2 = HELO, 3 = ARTY
*    FRVAL  - SORTED FORCE RATIO VALUES CORRESPONDING TO INDXFR
*    NUMSOR - NORMAL NUMBER OF COMBAT SUPPORT SYSTEMS PER MISSI
*    NOWALO - NUMBER OF SYSTEMS CURRENTLY AVAILABLE FOR ALLOCAT
*****************************************************
INCLUDE COMMONS
CASE: TYPE OF COMBAT SUPPORT
CASE 1, ALLOCATE CLOSE AIR SUPPORT
ARE MINIMUM CAS SORTIES AVAILABLE
   COPY ALLOCATION DATA FOR MESSAGE TO WOC
   CHANGE FORCE RATIO TO NUMBER SORTIES ALLOCATED
   SEND ALLOCATED MESSAGE TO WOC
CASE 2, ALLOCATE HELICOPTERS
ARE MINIMUM HELICOPTER SORTIES AVAILABLE
   SET NUMBER HELICOPTERS ALLOCATED
   DECREMENT NUMBER HELICOPTERS REMAINING TO ALLOCATE
   DECREMENT NUMBER AVAILABLE FOR TAKEOFF
   CREATE HELICOPTER ASSIGNED SUPPORT QUEUE ENTRY
   MAKE SUPPORT APPROVED MESSAGE
CASE 3, ALLOCATE CORPS ARTILLERY
ARE MINIMUM ARTILLERY TUBES AVAILABLE
   SET NUMBER TUBES ALLOCATED
   DECREMENT NUMBER TUBES REMAINING TO ALLOCATE
   DECREMENT NUMBER AVAILABLE IN THE AOC
   CREATE ARTILLERY ASSIGNED SUPPORT QUEUE ENTRY
   MAKE SUPPORT APPROVED MESSAGE
   PUT TUBES INTO COMBAT MATRIX
   COMBAT UNIT MATRIX IDENTIFIED
   INCREMENT TUBE COUNT IN SUMMARY
   RESET UNIT SUPPORT REQUEST TO NULL
```

```
SUBROUTINE CS_AVL(PNODE,PALLOC)
***************************************************
* REPRESENTS THE HOC GROUND OPERATIONS
*    PNODE  - INPUT CURRENT NODE
*    PALLOC - OUTPUT POINTER TO ALLOCATION BLOCK
***************************************************
INCLUDE COMMONS
INITIALIZE CURRENT AVAILABILITY TO ZERO
MAKE HELICOPTER READY QUEUE CURRENT
      REDUCE READY QUEUE
       GET NEXT HELICOPTER READY QUEUE
SET MAXIMUM HELICOPTERS AVAILABLE THIS TIME PERIOD
SET MAXIMUM ARTILLERY TUBES AVAILABLE THIS TIME PERIOD
      REDUCE READY QUEUE
       GET NEXT TUBE READY QUEUE
SET MAXIMUM TUBES AVAILABLE THIS TIME PERIOD
SET CAS SORTIES AVAILABLE THIS TIME PERIOD
       TEST FOR CORRECT SET OF ALLOCATION PARAMETERS
          NOT CORRECT SET OF ALLOCATION PARAMETERS, GET NEXT SE
    ALLOCATION PARAMETERS FOUND
    TEST FOR CURRENT ALLOCATION TIME PERIOD
    CALCULATE NUMBER OF SORTIES TO BE ALLOCATED
NUMBER OF SORTIES ALLOCATED
MINIMUM SORTIES ON A MISSION
MAXIMUM SORTIES TO ALLOCATE DURING TIME PERIOD
NUMBER OF SORTIES CURRENTLY AVAILABLE TO ALLOCATE
END OF CS_AVL
```

```
SUBROUTINE CS_DONE (PCMDR)
*******************************************************
*  WITHDRAWS ALL HELICOPTERS FROM MATRIX POSITION 12 AT
*     EACH TIME CYCLE AND WITH DRAWS ALL CORPS ARTILLERY
*     FROM POSITION 13 WHEN "SUPPORT NOT REQUIRED"
*     THREASHOLD IS MET.  HELICOPTERS ARE SCHEDULED TO
*     RETURN TO AVAILABLE QUEUE MINUS ENROUTE LOSSES.
*  ARTILLERY IS PUT DIRECTLY INTO CORPS ARTILLERY
*     AVAILABILITY LIST
*******************************************************
INCLUDE COMMONS
RETURN ALL COMMANDER'S HELICOPTERS FROM SUBORDINATES
      IF HELICOPTERS EXIST, RETURN THEM TO COMMANDER
         HELICOPTER READY TIME SET TO CURRENT TIME PLUS
         ENROUTE TIME PLUS TURN AROUND TIME
         NUMBER OF RETURNING HELICOPTERS IS DECREASED BY
            ENROUTE PS
      CHECK ON REQUIREMENT FOR ARTILLERY
      CHECK FOR CORPS ARTILLERY ASSIGNMENT
         USE COMMANDERS PERCEPTION TO PULL ARTILLERY
            CALCULATE TUBE AVAILABILITY TIME
      GET NEXT SUBORDINATE
LAST SUBORDINATE HANDLED
```

```
SUBROUTINE CS_OPS(PNODE,PALLOC)
*****************************************************
*  SEQUENCE ALLOCATION PROCESS BY CALLS TO CS_ALO
*  DELETE REQUESTS ON TEMPORARY QUEUE
*****************************************************
INCLUDE COMMONS
SET POINTER TO SUPPORT REQUESTS
   DO FOR EACH SUBORDINATE ON FQUEUE
      GET SUPPORT REQUEST
      INITIALIZE SUPPORT FLAG TO NONE PROVIDED
         SET WORKING FORCE RATIO
         SMALLEST FORCE RATIO SUPPORT
            SUPPORT WAS PROVIDED, TEST FOR ADDITIONAL
              TYPE TO BE PROVIDED
         SECOND LEVEL FORCE RATIO SUPPORT
         TEST FOR ESCALATION OF SUPPORT TYPE
         HIGHEST LEVEL FORCE RATION SUPPORT
      TEST FOR NO SUPPORT PROVIDED
         SEND NO SUPPORT MESSAGE
*** NOT IMPLEMENTED YET
    GET NEXT SUBORDINATE ON FORCE RATIO QUEUE
LOG SORTIES ALLOCATED
*** NOT IMPLEMENTED
RELEASE REQUEST QUEUE
```

```
 SUBROUTINE DMINIT(MEMORY,MPTR,IGBPTR,MAXDM)
*****************************************************
*   INITIALIZE DYNAMIC MEMORY
*****************************************************
*INITIALIZE MEMORY POINTER
*INITIAIZE GARBAGE POINTER
*CLEAR DYNAMIC MEMORY
```

```
      SUBROUTINE EIGENV(RB,BR,M,N,VR,VB,ALAM)
*****************************************************************
*THIS ROUTINE COMPUTES THE EIGEN VALUES AND LAMBDA FOR METH5 AND
*****************************************************************
      INCLUDE COMMONS
*
*     MAXIMUM NUMBER OF ITERATIONS TO FIND AN EIGENVECTOR IS MNIE
*     SMALLEST DIFFERENCE IN EIGENVALUES IS EFCE.
*     SMALLEST ALLOWABLE DENOMINATOR (TO AVOID DIVISION BY ZERO) IS EP
*
****************************************************************
```

```
SUBROUTINE ERROUT(MESAGE)
****************************************************
*  PRINT OUT ERROR CONDITIONS AND STOPS EXECUTION
****************************************************
INCLUDE COMMONS
```

```
SUBROUTINE EVENTS
*******************************************************
*  PROCESS SIMULATION EVENTS FROM CURRENT TIME TO END
*     OF RUN TIME
*  CALLS MSGIN, NODE, COMBAT, AIRBAS, TIMOUT
*******************************************************
 INCLUDE COMMONS
 INITIALIZE CMDRS STATUS BLOCK
 INITIALIZE TIME T OUTPUT IF REQUESTED
 START OF EVENTS IN A TIME INCREMENT
    PROCESS TIME T INPUT
    PROCESS EXTERNAL MESSAGES
    PROCESS NETWORK
    PROCESS AIRBASE
    PROCESS COMBAT
    CREATE OUTPUT
    PRODUCE TIME T OUTPUT IF REQUIRED
DAILY GRAPHIC OUTPUT FOR POSTPROC
TEST FOR END OF DAY
     MAKE NEXT EXTENT OF C3SUM
     MAKE NEXT EXTENT OF TIMET
     MAKE NEXT EXTENT OF LOSST
     PROCESS ALL LOSSES ON LOSSUM QUEUE
    TEST FOR LAST TIME INCREMENT
 INCREMENT GAME TIME
 END OF GAME TIME
```

```
SUBROUTINE EXTMSG
*****************************************************
*  GET MESSAGE FROM USER INPUT FOR CURRENT TIME
*  PUT MESSAGE ON ORIGINATORS FUTURE QUEUE
*  INPUT FOR MESSAGE
*    SEND TIME
*    MESSAGE TYPE
*    ORIGINATOR UNIT TYPE
*    DESTINATION UNIT ID
*    MESSAGE CREATION TIME
*    TRACKING FLAG
*    ATO FLAG
*    MESSAGE PRIORITY
*    TIME IN NETWORK
*  **  ATO DATA
*        COMBAT UNIT ID
*        EARLIEST TIME ON TARGET
*        LATEST TIME ON TARGET
*        AIRCRAFT TYPE
*        NUMBER OF AIRCRAFT
*        TYPE REQUEST, 0-PREPLAN, 1-IMMEDIATE
*****************************************************
INCLUDE COMMONS
SKIP OVER HEADER DOCUMENTATION
GET FIRST MESSAGE
CHECK FOR CURRENT TIME
   PROCESS THIS MESSAGE
   CHECK FOR ADDITIONAL DATA
      GET ADDITIONAL DATA
      END ADDITIONAL DATA
   FIND NODE OF DESTINATION
   PUT ON INPUT QUEUE
   GET NEXT EXTERNAL MESSAGE
END PROCESSING THIS TIME FRAME
```

```
SUBROUTINE EXTSPT(PMSG,PDATA)
*******************************************************
*  EXTSPT GETS THE DATA SECTIONS OF EXTERNAL MESSAGES 3136
*  AND PLACES THEM INTO THE DATA POINTER OF THE MESSAGE
*******************************************************
INCLUDE COMMONS
TEST FOR MESSAGE TYPE
GET 3126 OR 3130 INTEL REPORT
   END OF 3126 & 3130 INPUT
GET FIRST INTEL REPORT
GET ADDITIONAL DATA ELEMENTS OF THE REPORT
```

```
SUBROUTINE FIND(PIN, N, ID, POUT)
********************************************************
*  FIND A POINTER IN A QUEUE
********************************************************
*  INPUT
*     PIN - POINTER TO TOP OF QUEUE TO BE SEARCHED
*     N   - OFFSET FROM PIN TO COMPARE
*     ID  - VALUE TO MATCH WITH N
********************************************************
*  CREATES
*     POUT - POINTER TO DESIRED ELEMENT
********************************************************
INCLUDE COMMONS
DO FOR ALL QUEUE ELEMENTS
   COMPARE VALUES
GET NEXT ELEMENT
END OF SEARCH
```

```
SUBROUTINE FQUEUE(PNODE)
********************************************************
*  FQUEUE ORDERS THE SUBORDINATE'S RED/BLUE FORCE RATIO
*  QUEUE BY DESCENDING ORDER. THE QUEUE ROOT IS AT NODE+21
********************************************************
*   INPUT
*      PNODE - POINTER TO NODE
********************************************************
INCLUDE COMMONS
GET STATUS QUEUE
SET ALL FRQ POINTERS TO 5
FIND MAX FORCE RATIO
   GET NEXT PSTAT
CHECK IF MAX VALUE FOUND
   SET ENTRY AT END OF QUEUE
END OF ORDERING FORCE RATIO QUEUE
```

```
SUBROUTINE GIMME(NPTR, LEN, ISPACE)
*********************************************************
*  PROVIDE A BLOCK OF STORAGE FROM DYNAMIC MEMORY
*********************************************************
*  INPUT
*    LEN    - LENGTH OF BLOCK
*    ISPACE - ARRAY THAT CONTAINS DYNAMIC MEMORY
*********************************************************
*  OUTPUT
*    NPTR   - POINTER TO START OF ALLOCATED BLOCK
*********************************************************
*SEGMENT GET VIRTUAL SPACE
*INCLUDE COMMONS
*IOUT IS OUTPUT DEVICE
*SEARCH GARBAGE LIST
*DO UNTIL LIST ENDS
     *IF (SIZE .EQ. LENGTH) THEN
       *SET PTR TO FIRST BLOCK
       *SNAP GARBAGE PTR
*ALLOCATE VIRGIN STORAGE
*UPDATE VIR SPACE PTR
*STORAGE OVRFLOW
*ZERO SPACE BLOCK
*END SEGMENT
```

```
SUBROUTINE HOLDQ1 (IPASS)
*********************************************************
*  MOVE PRIORITY MESSAGES FROM HOLD QUEUE TO ALTERNATE
*  COMMUNICATIONS LINK1 SEND QUEUE
*********************************************************
*  CALLS - SNAP, FIND
*********************************************************
*  INPUT
*     IPASS - FLAG FOR SELECTION OF DESTINATION
*********************************************************
INCLUDE COMMONS
DO FOR ALL NODES
DO FOR ALL DESTINATIONS
SET ALLOCATION FLAG OFF
   DO FOR ALL LINKS
      DO FOR ALL MESSAGES ON HOLD QUEUE
         SET UP FOR MULTIPLE DESTINATIONS
            GET ALTERNATE DESTINATION
            CHECK IF ALTERNATE EXISTS
               FIND DESTINATION STRUCTURE
         GET FIRST ALT LINK OF MESSAGE
         CHECK FOR 0 TYPE
         GET ALTERNATE LINK
         COMPARE LINK TYPES
            LINK TYPES MATCH
            CHECK ALTERNATE LINK HOLD QUEUE PRIORITY
            CHECK SEND QUEUE
                  HOLD MESSAGE HAS GREATER PRIORITY
                  CHECK CAPACITY
                     PUT HOLD MESSAGE ON SEND QUEUE
                     SET FLAG = 1
               GET NEXT SEND MESSAGE
            END OF SEND QUEUE
            GET NEXT LINK
         GET NEXT MESSAGE ON HOLD QUEUE
         END OF HOLD QUEUE
      GET NEXT LINK
      END OF FIRST LINK LOOP

      MOVE EXCESS MESSAGES FROM SEND TO HOLD QUEUES
      DO FOR ALL LINKS
         DO UNTIL LINK CAPACTITY USED
            GET CORRECT CAPACITY
               GET NEXT MESSAGE
         MOVE EXCESS MESSAGES TO HOLD QUEUES
            CHECK FOR ALTERNATE STATUS
               PLACE IN THIS LINK HOLD QUEUE
               END THIS MOVE
            PLACE IN ANOTHER QUEUE
            GET NEXT MESSAGE
         END MESSAGE MOVES
      GET NEXT LINK
   END OF LINK QUEUE
GET NEXT DESTINATION
END OF DESTINATION QUEUE
GET NEXT NODE
LAST NODE COMPLETED
```

```
SUBROUTINE HOLDQ2 (IPASS)
********************************************************
*  MOVE PRIORITY MESSAGES FROM HOLD QUEUE TO ALTERNATE
*  COMMUNICATIONS LINK2 SEND QUEUE
********************************************************
*  CALLS - SNAP, FIND
********************************************************
*  INPUT
*     IPASS - FLAG FOR ALTERNATE DESTINATIONS
********************************************************
INCLUDE COMMONS
DO FOR ALL NODES
DO FOR ALL DESTINATIONS
SET ALLOCATION FLAG OFF
   DO FOR ALL LINKS
      DO FOR ALL MESSAGES ON HOLD QUEUE
         SET UP FOR MULTIPLE DESTINATIONS
            GET ALTERNATE DESTINATION
            CHECK IF ALTERNATE EXISTS
               FIND DESTINATION STRUCTURE
         GET SECOND ALT LINK OF MESSAGE
         CHECK FOR 0 TYPE
         GET ALTERNATE LINK
         COMPARE LINK TYPES
            LINK TYPES MATCH
            CHECK ALTERNATE LINK HOLD QUEUE PRIORITY
            LINK TYPES MATCH, CHECK SEND QUEUE
                  HOLD MESSAGE HAS GREATER PRIORITY
                  CHECK CAPACITY
                     PUT HOLD MESSAGE ON SEND QUEUE
                     SET FLAG = 2
               GET NEXT SEND MESSAGE
            END OF SEND QUEUE
            GET NEXT LINK
         GET NEXT MESSAGE ON HOLD QUEUE
         END OF HOLD QUEUE
      GET NEXT LINK
      END OF FIRST LINK LOOP

      MOVE EXCESS MESSAGES FROM SEND TO HOLD QUEUES
      DO FOR ALL LINKS
         DO UNTIL LINK CAPACITY USED
            GET CORRECT CAPACITY
               GET NEXT MESSAGE
         MOVE EXCESS MESSAGES TO HOLD QUEUES
            CHECK FOR ALTERNATE STATUS
               PLACE IN THIS LINK HOLD QUEUE
               END THIS MOVE
            PLACE IN ANOTHER QUEUE
            GET NEXT MESSAGE
         END MESSAGE MOVES
      GET NEXT LINK
   END OF LINK QUEUE
GET NEXT DESTINATION
END OF DESTINATION QUEUE
GET NEXT NODE
LAST NODE COMPLETED
```

```
SUBROUTINE INPUT
******************************************************
*  WORKS FOR T = 1 ONLY
******************************************************
*  CALLS FIND, GIMME
******************************************************
INCLUDE COMMONS
   CHECK FOR TIME T DATA
      PUT TIME T DATA ON UNIT 11
NODE IN
CHECK IF NODE STRUCTURE EXISTS
   CREATE NEW NODE STRUCTURE
   INITIALIZE NEW NODE STRUCTURE
   CREATE MESSAGE COUNT STRUCTURE
   CREATE RANDOM DISTRIBUTION TYPE STRUCTURE
CREATE DESTINATION STRUCTURE
CREATE STATUS STRUCTURE FOR SUBORDINATES
   SET COMMANDER ID
END NODE IN
   INPUT NODE MESSAGE PROCESSING LIMITS
   CHECK FOR TIME T DATA
      PUT TIME T DATA ON UNIT 12
   INITIALIZE DISTRIBUTION TYPE
   CHECK FOR TIME T DATA
      PUT TIME T DATA ON UNIT 13
LINK IN
PROCESS FOR EACH E D OF LINK
FIND ORIGIN
FIND DESTINATION
CREATE LINK STRUCTURE
PUT LINK ON END OF QUEUE
   END OF LINK QUEUE FOUND
INITIALIZE LINK DATA
CHECK FOR SECOND PASS
SWAP NODES AND REPEAT PROCESS
END OF LINK IN
INITIALIZE RANDOM DISTRIBUTION TYPES
RDIST IN
INPUT COMBAT DATA
INPUT AIR OPERATIONS DATA
INITIALIZE ALL ALTERNATE DESTINATION POINTERS, DESTINATION
  UNIT TYPES, AND POINTER TO COMMANDERS

DO FOR ALL NODES
   REPLACE COMMANDER UNIT NUMBER WITH POINTER
   DO FOR ALL DESTINATIONS
      FILL IN DESTINATION UNIT TYPE
      REPLACE DESTINATION ALTERNATES WITH POINTERS
      CHECK ALL DESTINATIONS FOR BOTH ALTERNATES
         SET ALTERNATE 1
      SET ALTERNATE 2
      GET NEXT DESTINATION
   GET NEXT NODE
LAST NODE CHANGED
PREPARE TIME T FILES FOR USE
```

```
SUBROUTINE INPUTA
*****************************************************
*  INPUT COMBAT SUPPORT OPERATIONS VALUES
*  ITYPE - TYPE OF INPUT LINE (CRC,'   ',END)
*  NODEC - CRC UNIT NUMBER
*  ATIME - ALERT RESPONSE TIME
*  ETIME - TIME FROM TAKE OFF TO TARGET
*  MINAC - MINIMUM NUMBER OF AIRCRAFT ON MISSION
*  PS    - PROBABILITY OF SURVIVAL OF CAS AIRCRAFT
*  NODEW - WOC UNIT NUMBER
*  ACTYPE - AIRCRAFT TYPE
*  ACTIME - TIME AIRCRAFT READY FOR TAKE OFF
*  NUMAC - NUMBER AIRCRAFT READY FOR TAKE OFF
*  TUBETYPE-ARTILLERY TUBE TYPE
*  TUBETIME-TIME ARTILLERY READY FOR ASSIGNMENT
*  NTUBES - NUMBER ARTILLERY TUBES READY FOR ASSIGNMENT
*  COMMENT - DOCUMENTATION AND REVIEW
****************************************************
INCLUDE COMMONS
READ IN THE INPUTS AND PRINT THEM OUTMAP
READ IN THE PREAMBLE DOCUMENTATION
READ IN HEADER DOCUMENTATION
READ WAIT TIME BETWEEN REQUESTS FOR SUPPORT & DISTRIBUTIONS
READ IN HEADER DOCUMENTATION FOR ALLOCATION FACTORS
   FIND NODE
   CREATE STRUCTURE FOR ALLOC
   STORE PARAMETERS IN ALLOC
READ IN THE HEADER DOCUMENTATION FOR CRC FACTORS
GET NEXT DATA LINE
   CREATE CRC
CREATE THE HEADER STRUCTURE FOR AIRCRAFT AVAILABILITY
READ HEADER FOR AIRCRAFT AVAILABILITY
INPUT AIRCRAFT FACTORS
   READY STRUCTURE
   GET NEXT AIRCRAFT ENTRY
   GET AIRCRAFT READY BLOCK
      MAKE NEW READY BLOCK
      CREATE RDYQ BLOCK
   END OF INPUT FOR AIR OPERATIONS
HELICOPTER SUPPORT
HELO SUPPORT, GET OPS DATA
CREATE SUPPORT OPS IN NODE BLOCK
   GET NEXT HELICOPTER ENTRY
   READY STRUCTURE
   GET NEXT HELICOPTER ENTRY
   GET AIRCRAFT READY BLOCK
      MAKE NEW READY BLOCK
      CREATE RDYQ BLOCK
ARTY SUPPORT, GET OPS DATA
CREATE SUPPORT OPS IN NODE BLOCK
   GET NEXT ARTILLERY ENTRY
   READY STRUCTURE
   GET NEXT ARTILLERY ENTRY
   GET ARTILLERY READY BLOCK
      MAKE NEW READY BLOCK
      CREATE RDYQ BLOCK
DEBUG PRINT OF CRC AND WOC STRUCTURES
SET NORMAL NUMBER OF SORTIES/MISSION FOR ALLOCATION
SET CAS
SET HELICOPTERS
SET ARTILLERY TUBES
```

```
SUBROUTINE INPUTC
*****************************************************
*  INPUT COMBAT VALUES
*    FRBCAS - FORCE RATIO, RED/BLUE LIMIT FOR CAS SUPPORT
*    FRBHEL - FORCE RATIO, RED/BLUE LIMIT FOR HELICOPTER SUPPOR
*    FRBART - FORCE RATIO, RED/BLUE LIMIT FOR ARTILLERY SUPPORT
*    FRBMIN - FORCE RATIO MINIMUM OF CAS, HELO AND ARTY
*    FRBVAR - VARANCE AROUND FRBART
*    INCLFR - WEAPONS TO BE USED IN FRATIO CALCULATION=1
*    NODE   - UNIT NUMBER
*    PENGR,PENGB - ENGAGEMENT RATE OF UNIT,0 = RESERVE,
*                                     ELSE = RATE INDEX
*    KOPSR,KPOSB - UNIT POSTURE INDEX (1-3)
*    NR,NB  - NUMBER OF COMBAT SYSTEMS
*    V      - INITIALIZED TO 1. FOR EIGEN VALUE BEST GUESS
*    WR,WB  - FORCE RATIO WEIGHTS FOR RED AND BLUE
*    INDXFR - INDEX ARRAY SORTED BY INCREASING FORCE RATIO FOR
*             COMBAT SUPPORT TYPE 1 = CAS, 2 = HELO, 3 = ARTY
*    FRVAL  - SORTED FORCE RATIO VALUES CORRESPONDING TO INDXFR
*    NUMSOR - NUMBER OF COMBAT SUPPORT SYSTEMS ALLOCATED PER MI
*****************************************************
INCLUDE COMMONS
READ IN THE INPUTS AND PRINT THEM OUT
READ IN PREAMBLE DOCUMENTATION
READ IN GENERIC RED UNIT DATA
READ IN FORCE RATIO THREASHOLD DATA VALUES
READ FORCE RATIO LIMITS FOR RED AND BLUE POSTURE CONTROL
READ IN RED FORCE RATIO WEIGHTS
READ IN BLUE FORCE RATIO WEIGHTS
INITIALIZE INDXFR AND RELATED ARRAYS
```

```
SUBROUTINE INTLUP (PNODE,POMP,POUT)
*****************************************************
*  SUBROUTINE INTLUP UPDATES COMMANDERS PERSEPTIONS OF
*  SUBORDINATES COMBAT FOES VIA MESSAGE 3136
*****************************************************
*  INPUT
*    PNODE  - POINTER TO NODE
*    POMP   - POINTER TO OUTPUT TYPE
*    POUT   - POINTER TO OUTPUT MESSAGE TYPE
*****************************************************
INCLUDE COMMONS
GET FIRST SPOT REPORT
IF REPORT MATCHES PROCESS, LOG DATA
      TEST FOR SUBORDINATE STATUS STRUCTURE
  FIND SUBORDINATE'S STATUS STRUCTURE
     ERROR, NO SUBORDINATE STATUS STRUCTURE
      UPDATE EXISTING FOE ESTIMATE IF NEW DATA IS LATEST
      CHECK DATA CURRENCY
         DELETE CURRENT ESTIMATES
      ENTER NEW ESTIMATES
  OLD INTEL DATA RECEIVED
      DELETE THESE DATA STRUCTURES
  GET NEXT SPOT REPORT
    ALL SPOT REPORTS PROCESSED FOR THIS RULE
```

```
SUBROUTINE KILLS(EX,PKXY,ALLXY,J,I,EAKXY)
**********************************************************
*  THIS ROUTINE COMPUTES THE KILL RATE MATRICES
**********************************************************
*     THE RATE WEAPON X KILLS WEAPON Y IS A PRODUCT OF ENGAGEMENTS,ALL
*     AND PROBABILITY OF KILL FOR THOSE WEAPONS.
```

```
SUBROUTINE LIMIT
**************************************************
* LIMITS THE MESSAGE TRAFFIC ON EACH LINK TO THE TWO WAY MAXIM
* CAPACITY
**************************************************
INCLUDE COMMONS
DO FOR ALL NODES
   DO FOR ALL DESTINATIONS
      SET LINK RECEIVER
      SET RECEIVER'S DESTINATION
      TEST IF LIMITING HAS BEEN DONE FOR THIS DESTINATION
      DO FOR ALL LINKS
         INITIALIZE COUNTERS
         FIND RECEIVERS LINK
         DO UNTIL CAPACITY USED
         CHECK PRIORITY OF BOTH MESSAGES
            TOP OF SEND LOOP
               GET CAPACITY FOR MESSAGE
               CHECK OTHER END OF LOOP
                  GET CAPACITY FOR MESSAGE
               BOTH QUEUES ENDED IN CAPACITY
            LINK CAPACITY EXCEDDED
            CORRECT LINK CAPACITY USED
            MARK THE END OF BOTH SEND QUEUES
            PROCESS BOTH QUEUES TO HOLD QUEUES
            MOVE EXCESS MESSAGES TO HOLD QUEUES
               PRIORITY CHECK
                  PRIORITIES SAME, CHECK CAPACITY
                     PUT MESSAGE BACK ON SEND QUEUE
                  END OF CAPACITY RECHECK
               END OF EQUAL PRIORITY CHECK
               PLACE IN APPROPRIATE QUEUE
               FIND DESTINATION
               FIND LINK
               RESET FLAG
               GET NEXT MESSAGE
            END MESSAGE MOVES
            INITIALIZE FOR SECOND PASS
      GET NEXT LINK
   GET NEXT DESTINATION
   FLAG BOTH DESTINATION QUEUES AS COMPLETED
GET NEXT NODE
LAST NODE
```

```
SUBROUTINE LOSOUT
************************************************************
*  PRINT CUMULATIVE LOSSES ON C3SUM FILE
*  CALLED BY C3EVAL AT END OF RUN
************************************************************
INCLUDE COMMONS
```

```
SUBROUTINE MAKMSG(MTYPE,ISEND,PNODE,IDEST,L1,L2,L3,ICAP,
*****************************************************
* MAKE SINGLE UNIT DESTINATION MESSAGES
* INPUT
*    MTYPE   MESSAGE NUMBER
*    ISEND   UNIT TYPE OF ORIGINATOR
*    PNODE   POINTER TO SENDER'S NODE
*    IDEST   DESTINATION UNIT
*    L1-L3   COMMUNICATION LINK TYPES
*    ICAP    COMMUNICATION CAPACITY REQUIRED
*    PATO    POINTER TO AIR TASKING ORDER
*    IORIG   ORIGINATOR UNIT (98 IS INTERNAL)
* OUTPUT
*    PMSG    POINTER TO MESSAGE
* ****************************************
* OTHER MESSAGE ELEMENTS SET
*    CREATE TIME TO NOW
*    MAXIMUM AGE TO 3
*    PRIORITY TO 1
*    OUTPUT FLAG IS ON
*****************************************************
INCLUDE COMMONS
TEST RANDOM OPERATION
   GET RANDOM DELAY
       GET RANDOM MESSAGE LENGTH
   IF NO RETURN NODES, USE SUPPORT UNIT
GET ALTERNATES FOR MESSAGE
```

```
SUBROUTINE MAP
*****************************************************************
*  SUBROUTINE MAP COMPUTES THE FORCE RATIOS BETWEEN RED AND BLUE FORC
*    DESCRIPTION OF INPUTS FOR RED FORCES ARE
*       NTRJ=NUMBER OF TYPES OF WEAPONS-RED
*       NAMER(J)=NAMES ASSOCIATED WITH EACH OF THE NTRJ WEAPONS
*       NTCR(J)=NUMBER OF TYPE-J RED WEAPONS (TYPICAL CASE)
*       NR(J)=NUMBER OF TYPE-J RED WEAPONS (ACTUAL)
*       ER(J)=ENGAGEMENTS, PER TIME PERIOD, FOR EACH RED TYPE-J WEAPO
*       ALTCRB(J,I)=ALLOCATION (TYPICAL-CASE) RED AGAINST BLUE
*       PKRB(J,I)=PROBABILITY THAT RED J KILLS BLUE I GIVEN ENGAGEMEN
*    DESCRIPTION OF INPUTS FOR BLUE FORCES ARE
*       NTBI=NUMBER OF TYPES OF WEAPONS-BLUE
*       NAMEB(I)=NAMES ASSOCIATED WITH EACH OF THE NTBI WEAPONS
*       NTCB(I)=NUMBER OF TYPE-I BLUE WEAPONS (TYPICAL CASE)
*       NB(I)=NUMBER OF TYPE-I BLUE WEAPONS (ACTUAL)
*       EB(I)=ENGAGEMENTS, PER TIME PERIOD, FOR EACH BLUE TYPE-I WEAP
*       ALTCBR(I,J)=ALLOCATION (TYPICAL-CASE) BLUE AGAINST RED
*       PKBR(I,J)=PROBABILITY THAT BLUE I KILLS RED J GIVEN ENGAGEMEN
*       THE ALGORITHM IS SET UP TO ACCEPT NO MORE THAT  11  DIFFERENT
*       TYPES.
*****************************************************************
      INCLUDE COMMONS
*     ALLOCATION IS COMPUTED TWICE, RED AGAINST BLUE AND BLUE AGAINS
*    COMPUTE THE KILL RATE MATRICES (ACTUAL CASE)
      SET ENGAGEMENT RATES
*    COMPUTE K (ACTUAL CASE)
*    OUTPUT THE INPUTS
```

```
SUBROUTINE MDELAY(PNODE,PFMQ,PDEST,JP1,JP2,JC1,JC2,
****************************************************************
* DETRMINES WHICH MESSAGE WILL BE SENT, HELD (SET SEND TIME TO
* NEXT TIME) OR DELETED. PROCESSES HOLD QUEUE ON EACH LINK FOR
* THE DESTINATION.
*    CALLED BY SUBROUTINE MOULIM
*    PARAMETERS
*      PDEST - POINTER TO DESTINATION STRUCTURE
*      JP1   - MAXIMUM PRIORITY MESSAGE PROCESSED
*      JC1   - NUMBER OF MESSAGES AT LEVEL JP1 PROCESSED
*      JP2   - MINIMUM PRIORITY MESSAGE DELETED
*      JC2   - NUMBER OF MESSAGES AT LEVEL JP2 NOT DELETED
*      NHOLD - NUMBER OF MESSAGES HELD FOR FUTURE PROCESSING
*      NDEL  - NUMBER OF MESSAGES DELETED
*      INDEX - =0, PROCESS FIRST DESTINATION FOR A NODE
*              =1, PROCESS ADDITIONAL DESTINATION
****************************************************************
INCLUDE COMMONS
PROCESS FIRST DESTINATION FOR A NODE
DO EACH LINK
      PRIORITY MESSAGE TO BE SENT
         IGNORE ALTERNATE MESSAGES
            IGNORE FUTURE MESSAGES
               TEST FOR EQUAL PRIORITY
                  TEST ENOUGH MESSAGES AT LOWER PRIORITY
               TEST FOR MESSAGE LESS THAN HIGHER PRIORITY
                     DELAY THIS MESSAGE
                     MOVE TO FMQ
               DELETE THIS MESSAGE
               TEST FOR ADDITIONAL DATA
         INCREMENT BACK POINTER
         GET NEXT MESSAGE
```

```
SUBROUTINE MINLIM(PNODE,MFLAG)
******************************************************
*  SUBROUTINE MINLIM LIMITS THE NUMBER OF MESSAGES THAT MAY
*  ENTER A NODE AT ANY ONE TIME
******************************************************
INCLUDE COMMONS
ZERO OUT MESSAGE IN COUNTERS
COUNT MESSAGS BY PRIORITY
   DON'T COUNT ALTERNATE MESSAGES
   DETERMINE CUT OFF PRIORITIES
      REMOVE MESSAGES FROM IMQ BY LOW PRIORITY
         DO NOT MOVE ALTERNATE MESSAGES
         DO NOT MOVE MESSAGE WITH ACCEPTABLE PRIORITY
            REMOVE THIS MESSAGE
            CHECK MESSAGE OUT OF AGE
            CHECK FOR MESSAGE PRIORITY BELOW DELETE LEVEL
               DELETE THIS MESSAGE
               CHECK FOR MEAASGE DATA
                  RETURN DATA SPACE
               RELEASE MESSAGE SPACE
            PUT MESSAGE ON HOLD QUEUE
            NEXT MESSAGE WHEN KEEPING MESSAGE ON IMQ
            GET NEXT MESSAGE
         END OF IMQ
RETURN MESSAGES ON HOLD TO IMQ
```

```
SUBROUTINE MOULIM
************************************************************
* LIMITS GENERATION OF OUTPUT MESSAGES AT EACH NODE BASED ON
* THE NUMBER OF MESSAGES REQUIRED TO BE PROCESSED AT THIS NODE
* AND MESSAGE PRIORITY.
*      CALLED BY:  SUBROUTINE NODE AFTER PROCESSING DECISION
*                  RULES BUT BEFORE MESSAGE PATH ALLOCATION.
************************************************************
INCLUDE COMMONS
DO FOR ALL NODES
     DO FOR EACH MESSAGE ON FUTURE QUEUE
        CHECK TIME
              CHECK AGE OF MESSAGE
                 DELETE THIS MESSAGE
                    DELETE DATA STRUCTURE
                 DELETE MESSAGE STRUCTURE
              GET NEXT MESSAGE
           MOVE MESSAGE TO HOLD QUEUE
              ERROR IN ROUTING
           DESTINATION FOUND
           FIND LINK TYPE
              ERROR IN ROUTING
     GET NEXT MESSAGE
   CLEAR COUNTER ARRAY
   DO FOR ALL DESTINATIONS
      DO FOR ALL LINKS
         COUNT APPLICABLE MESSAGES ON HOLD QUEUE
            DON'T COUNT ALTERNATE MESSAGES
            DON'T COUNT MESSAGES TO BE SENT IN THE FUTURE
         COUNT APPLICABLE MESSAGES ON SEND QUEUE AND MOVE ALL
            MESSAGES FROM IT TO THE HOLD QUEUE
            DON'T COUNT ALTERNATE MESSAGES
             IF(MEMORY(PNODE+1).EQ.12)
            DON'T COUNT MESSAGES TO BE SENT IN THE FUTURE
   DETERMINE CUT OFF PRIORITIES
      DELAY OR DELETE MESSAGES
      COMMANDER AS DESTINATION FIRST
      NEXT PROCESS SUBORDINATE DESTINATIONS
      FINALLY PROCESS OTHER DESTINATIONS
   TEST FOR OUTPUT
   GET NEXT NODE
```

```
SUBROUTINE MOVMSG(PMSG, PMSGO, PDEST, IUNIT, IALT)
*****************************************************
*  MESSAGES IN THE PMSG QUEUE ARE COMPARED TO DESTINATION
*  UNIT TYPE AND TO MESSAGES IN THE PDEST > PSEND QUEUE
*  FOR LINK TYPE, PRIORITY AND CAPACITY.  APPROPRIATE
*  MESSAGES ARE MOVED AND LOWER PRIORITY MESSAGES ON THE
*  SEND QUEUE WILL BE BUMPED.
*****************************************************
*  INPUT
*     PMSG -  POINTER TO SOURCE QUEUE
*     PMSGO -  LOCATION OF PMSG
*     PDEST -  POINTER TO A DESTINATION STRUCTURE
*     IUNIT -  DESTINATION UNIT
*     IALT  -  ALTERNATE DESTINATION FLAG
*           -  FIRST ALT = 1,  2ND ALT = 2
*****************************************************
INCLUDE COMMONS
SET ALTERNATE FLAG
DO FOR ALL MESSAGES
   CHECK MESSAGE FOR NO ALTERNATE
      CHECK FOR DESTINATION UNIT
         UNIT OK, TRY EACH LINK
            GET LINK
            LINK TYPE MATCHES
            PUT MESSAGE ON SEND QUEUE BY PRIORITY
                  CHECK CAPACITY
                     PUT MESSAGE ON SEND QUEUE
                     RESET TRANSMISSION FLAG
               GET NEXT MESSAGE ON SEND QUEUE
               LINKS DONT MATCH, GET NEXT LINK
         END LINK QUEUE
      UNIT TYPE WRONG
   LAST DESTINATION
GET NEXT MESSAGE ON MESSAGE QUEUE
LAST MESSAGE PROCESSED
```

```
SUBROUTINE MPROD(A,B,N,M,L,R)
*****************************************************************
*  THIS SUBROUTINE MULTIPLIES TWO MATRICES.
*****************************************************************
```

```
SUBROUTINE MSGBAD(PMSG,PNODE,NOMORE)
*****************************************************
*  PROCESS CHANGES TO MESSAGES DUE TO PARTIAL MESSAGES
*     BEING RECEIVED OR COMMUNICATIONS SYSTEM CHANGES
*     TO THE CONTENTS OF MESSAGES THAT HAVE CONTENTS
*     (MESSAGE TYPES 3000, 3400, 7000, 3126, 3130)
*  DISTRIBUTIONS ARE A FUNCTION OF MESSAGE TYPE
*****************************************************
*  INPUT
*     PMSG - POINTER TO MESSAGE
*     PNODE - POINTER TO PROCESSING NODE
*****************************************************
*  OUTPUT
*     NOMORE - FLAG OF PROCESS STATUS
*              IF SET TO 0, MESSAGE IS STILL IN THE
*                MESSAGE QUEUE
*****************************************************
INCLUDE COMMONS
GET RANDOM NUMBER DISTRIBUTION
***   PARTIAL MESSAGE PROCESSING ***
    REQUEST MESSAGE BE RESENT
    INCREMENT RESEND COUNTER
    PUT MESSAGE ON FUTURE QUEUE
 DELETE GARBLED MESSAGE
    DELETE MESSAGE DATA SEGMENT
 INCREMENT GARBLED MESSAGE COUNTER
 *** MESSAGE CONTENT MODIFICATION
        DETERMINE MESSAGE TYPE
        *** ATO PROCESS
        *** SPOT REPORT PROCESS
 END PROCESSING MSGBAD
```

```
SUBROUTINE MSGIN(PMSG, POMP, PNODE)
****************************************************
*  MSGIN LOGS AN INPUT MESSAGE INTO EACH OUTPUT TYPE BY
*  ORIGINATOR AND INPUT TYPE, SAVES MESSAGE DATA AND
*  DELETES MESSAGE SPACE
****************************************************
*  CALLS GIMME, RELEAS
****************************************************
*  INPUT
*     PMSG - POINTER TO INPUT MESSAGE
*     POMP - POINTER TO DESTINATION NODE
*            OUTPUT MESSAGE TYPE QUEUE
****************************************************
INCLUDE COMMONS
GET INPUT MESSAGE TYPE
SPECIAL PROCESSING FOR RESEND REQUESTS
GET INPUT MESSAGE ORIGINATOR AND TYPE
DO FOR ALL OUTPUT TYPES
   GET FIRST INPUT POINTER
   DO UNTIL INPUT TYPE FOUND
      TEST MESSAGE TYPE
         TEST ORIGINATOR TYPE UNIT
            LOOK FOR EXISTING UNIT LOG
            GET NEXT LOG
            ORIGINATOR NOT ON LIST, CREATE ENTRY
        PUT RULE NUMBER IN DATA STRUCTURE
            SET FLAG AND AGE
            CHECK FOR OTIME; IF OLD END PROCESS
            SET MESSAGE TRACK FLAG
   NEXT INPUT TYPE
NEXT OUTPUT TYPE
LAST POMP, SAVE MESSAGE DATA
CASE (MESSAGE TYPE)
   AIR OPERATIONS
      PUT ATO ON ORDER QUEUE
      PUT ATO ON REQUEST QUEUE
         SET RETURN NODE POINTER
SPOT LOSS REPORT
END MSGIN
```

```
SUBROUTINE MSGOUT (PNODE, POMP,POUT,PDATA,PLENTH)
*******************************************************
*  MSGOUT GENERATES AN OUTPUT MESSAGE TO ALL DESIGNATED
*  DESTINATIONS
*  THE CONTENT OF THE MESSAGE MAY BE RANDOMLY CHANGED
*  THE TIME TO SEND THE MESSAGE MAY BE RANDOM
*******************************************************
*  CALLS - GIMME, SNAP, FIND, ERROUT, RULES
*******************************************************
*  INPUT
*     PNODE - POINTER TO NODE
*     POMP  - POINTER TO OUTPUT TYPE
*     POUT  - POINTER TO OUTPUT MESSAGE TYPE
*     PDATA - POINTER TO MESSAGE DATA
*     PLENTH- LENGTH OF BLOCK FOR MESSAGE DATA
*******************************************************
INCLUDE COMMONS
   CHECK FOR COMMANDER ONLY
      COMMANDER ONLY, SET DESTINATION
   DO FOR EACH DESTINATION = DESTINATION TYPE
      CHECK DESTINATION TYPE
      GET NEXT DESTINATION
      DESTINATION FOUND
   CREATE OUTPUT MESSAGE
   ENTER DATA
   TEST FOR RANDOM CHANGE OF MESSAGE INDICATED
      GET DANDOM DELAY
      GET RANDOM MESSAGE LENGTH
   SET ALTERNATE DESTINATIONS
      SET FIRST ALTERNATE TO FIRST NODE ALTERNATE
   TRY SECOND ALTERNATE
      SET FIRST ALTERNATE TO SECOND NODE ALTERNATE
         SET SECOND ALTERNATE TO NODES FIRST ALTERNATE
   TRY SECOND MESSAGE ALTERNATE
      SET FIRST ALTERNATE TO FIRST NODE ALTERNATE
   TRY SECOND NODE ALTERNATE
      SET FIRST ALTERNATE TO SECOND NODE ALTERNATE
   TRY SECOND NODE ALTERNATE
      SET SECOND ALTERNATE TO SECOND NODE ALTERNATE
   END OF ALTERNATE LOGIC
   CHECK FOR MESSAGE TRACKING FLAG
      SET TRACKING FLAG
   CHECK FOR MESSAGE SEND TIME
      PUT MESSAGE ON FUTRUE QUEUE BY TIME
   PUT MESSAGE ON SEND QUEUE BY PRIORITY
      GET SEND QUEUE
      FIND LINK TYPE
   ATTACH MESSAGE BLOCK
   FINISHED IF COMMANDER ONLY
   GET NEXT DESTINATION
END OF OUTPUT MESSAGES
```

```
SUBROUTINE NODE
*********************************************************
*  PROCESS C3 EVENTS
*  CALLS - FIND, ERRCUT, SNAP, MSGIN, MSGOUT, ALOCAT,
*     ALTOUT, LIMIT, SEND
*********************************************************
INCLUDE COMMONS
PROCESS INPUT MESSAGES
OPTIONAL PRINT
 IF(PFLAG(20).EQ.0) GO TO 8
   ALTERNATE DESTINATIONS
   INPUT QUEUES
   GET FIRST NODE
   DO FOR ALL NODES
      GET NODE IDENT.
      LIMIT NUMBER OF INPUT MESSAGES
      DO FOR ALL MESSAGES ON INPUT QUEUE
         IF MESSAGE IS ADDRESSED TO ANOTHER NODE
            REROUTE MESSAGE, PUT ON HOLD QUEUE
               FIND DESTINATION LINK
               ERROR IN ROUTING
            DESTINATION FOUND, SNAP IN
            RESET ALTERNATE COMMUNICATION FLAG
            FIND LINK TYPE
            CHECK LINK EXISTANCE
               TRY FIRST ALTERNATE LINK
                  LINK MATCHES, MODIFY MESSAGE
               TRY SECOND ALTERNATE LINK
                  LINK MATCHES, MODIFY MESSAGE
                DELETE MESSAGE
         USE DECISION RULES FOR MESSAGE PROCESSING
            PARTIAL MESSAGE AND MESSAGE CONTENT RANDOM PROCESS
      GET NEXT MESSAGE
   LAST MESSAGE, GET NEXT NODE
   RETURN ANY HELD INPUT MESSAGES TO IMQUEUE
LAST NODE
PROCESS DECISION RULES
   CALL ATOLIST
   GET FIRST NODE
   DO FOR ALL NODES
      DO FOR ALL OUTPUT MESSAGE TYPES
      GET FIRST TYPE OUTPUT
         TEST FOR PERIODIC PROCESS
         CHECK FOR PERIODIC TIME
            TEST FOR FIRST TIME FOR PROCESS
         TEST FOR RANDOM PROCESS
         DO FOR ALL INPUT MESSAGE TYPES
         GET FIRST TYPE INPUT
         INITIALIZE FLAG SUM
            DO FOR ALL INPUT MESSAGES
            INCREMENT AGE
            TEST FOR SINGLE USE FLAG
               TEST FOR USE OF MESSAGE
            TEST AGE GREATER THAN LIMIT
               SET FLAG TO OLD
            SUM INPUT FLAGS
            TEST FOR EACH MESSAGE TO CREATE A PROCESS
            GET NEXT INPUT MESSAGE
            GET NEXT INPUT TYPE
         INPUT TYPES COMPLETE, TEST FOR OUTPUT
            OUTPUT ACTION REQUIRED
            OUTPUT ACTION IS PERIODIC
         GET NEXT OUTPUT TYPE
      GET NEXT NODE
```

```
LAST NODE
LIMIT MESSAGES PROCESSED
ALLOCATE OUTPUT TO LINKS
ADD ALTERNATE ROUTINGS
ADJUST LINKS TO LIMIT
SECOND ALTERNATE ROUTINGS
ADJUST LINKS TO LIMIT
FIRST ALTERNATE DESTINATION
   LINK2
SECOND ALTERNATE DESTINATION
   LINK 2
THIRD ALTERNATE ROUTING
CALCULATE SUPPORT OPERATIONS
OPTIONAL PRINT
   OUTPUT QUEUES
   FUTURE QUEUES
   HOLD QUEUES
SEND MESSAGES
END OF NODE PROCESSING
```

```
SUBROUTINE OUTPUT
*************************************************************
*  PRODUCES A SUMMARY OF MESSAGE TRAFFIC AND CLOSE AIR SUPPORT
*************************************************************
INCLUDE COMMONS
IOUT IS OUTPUT DEVICE
DO FOR ALL NODES
   DO FOR ALL DESTINATIONS
      DO FOR ALL LINKS
         DO FOR ALL MESSAGES ON HOLD
         GET NEXT MESSAGE
      GET NEXT LINK
   GET NEXT DESTINATION
GET NEXT NODE
LAST NODE
```

```
SUBROUTINE PMSG1(PF)
*******************************************************
* PRINT OUT ALTERNATE DESTINATION, INPUT AND FUTURE
*  MESSAGES UNDER PRINT FLAG CONTROL
*  INPUT
*     PF = 1 ALTERNATE DESTINATION
*          2 INPUT QUEUES
*          4 FUTURE QUEUES
*******************************************************
INCLUDE COMMONS
IOUT IS OUTPUT DEVICE
           PRINT ALL NODES
              TEST FOR ALTERNATE DESTINATION
            GET NEXT NODE
 TEST FOR ALL INPUT MESSAGES
   TEST FOR FIRST PRINT TIME
      TEST FOR LAST PRINT TIME
         TEST FOR SPECIFIC NODE PRINT
            PRINT ALL NODES
               TEST FOR ALTERNATE DESTINATION
            GET NEXT NODE
         PRINT SPECIFIC NODE
            TEST FOR ALTERNATE DESTINATION
 TEST FOR TRACKED MESSAGES
   PRINT TRACKED MESSAGES ONLY
      TEST FOR ALTERNATE DESTINATION
      TEST FOR TRACKING FLAG
      GET NEXT NODE
```

```
SUBROUTINE PMSG2(PF)
**********************************************************
* PRINT OUT ALTERNATE OUTPUT AND HOLD
*  MESSAGES UNDER PRINT FLAG CONTROL
*  INPUT
*     PF = 3 OUTPUT QUEUES
*          5 HOLD QUEUES
**********************************************************
INCLUDE COMMONS
IOUT IS OUTPUT DEVICE
TEST FOR ALL INPUT MESSAGES
  TEST FOR FIRST PRINT TIME
     TEST FOR LAST PRINT TIME
        TEST FOR SPECIFIC NODE PRINT
           PRINT ALL NODES
                 GET NEXT LINK
              GET NEXT DESTINATION
           GET NEXT NODE
        PRINT SPECIFIC NODE
                 GET NEXT LINK
              GET NEXT DESTINATION
TEST FOR TRACKED MESSAGES
  PRINT TRACKED MESSAGES ONLY
                 GET NEXT LINK
              GET NEXT DESTINATION
     GET NEXT NODE
```

```
      SUBROUTINE POSTURE(FRB,PCMBT)
*****************************************************************
*   SETS BLUE AND RED UNITS POSTURES AS A FUNCTION OF PARAMETER
*   FORCE RATIO AND FORCE RATIO LIMITS FROM INPUT.
*
*   FRB   - FORCE RATIO RED/BLUE
*   PCMBT - POINTER TO UNITS COMBAT DATA STRUCTURE
*****************************************************************
      INCLUDE COMMONS
*GET CURRENT POSTURES FROM CMBT DATA STRUCTURE
*TEST FOR AUTO POSTURE CHANGE OPERATION
```

```
SUBROUTINE POUT (NUM, LENGTH)
*************************************************************
*  SUBROUTINE POUT PRODUCES A SNAP SHOT OF PART OF DYNAMIC
*  MEMORY
*************************************************************
INCLUDE COMMONS
```

```
SUBROUTINE PRATIO(PNODE)
*****************************************************
*  PRATIO CALCULATES SUBORDINATES RED/BLUE FORCE RATIO
*  AND ENTERS IT INTO THEIR STATUS STRUCTURE
*****************************************************
*  INPUT
*     PNODE - POINTER TO NODE
*****************************************************
INCLUDE COMMONS
GET STATUS QUEUE
DO FOR EACH SUBORDINATE STATUS STRUCTURE
   INITIALIZE RED FORCE ARRAY
   SUM PERCIEVED RED FORCES
   DO FOR EACH FOE
   TEST FOR FOE FORCES
      SET FORCE RATIO TO ZERO
   CALCULATE CURRENT FORCES
   FORCE RATIO BASED ON THE FIRST 11 SYSTEMS ONLY
   CALCULATE FORCE RATIO
   SET FORCE RATIO CALCULATION TIME
   GET NEXT STATUS STRUCTURE
FINISHED FORCE RATIO CALCULATIONS
```

```
      SUBROUTINE PRNT(NTX,NTY,NTCX,NX,NAMEX,NAMEY,VALX,PKXY,EX,
     XALTCXY,ALLXY,WGTX,BCPX,K2,VULFX, CAXW,IND)
      ************************************************************
      * THIS ROUTINE OUTPUTS HEADINGS AND CONTROLS THE PRINTING OF THE
      *    VARIOUS VARIABLES
      ************************************************************
      INCLUDE COMMONS
      OUTPUT NUMBER OF WEAPON TYPES
      OUTPUT NUMBER OF EACH TYPE OF WEAPON
      OUTPUT THE TYPICAL AND ACTUAL CASE NUMBERS
      OUTPUT THE AVERAGE NUMBER OF ENGAGEMENTS
      OUTPUT TYPICAL CASE ALLOCATION
      OUTPUT ACTUAL CASE ALLOCATION
      OUTPUT THE PROBABILITIES OF KILL
```

```
      SUBROUTINE PRNT2(NAMEX,PARAM1,PARAM2,NT,FMT1,FMT2,TWO)
*************************************************************
*  THIS ROUTINE IS USED TO PRINT 1 DIMENSIONAL ARRAYS
*************************************************************
      INCLUDE COMMONS
      LUPLIM IS THE TOTAL NUMBER OF ROWS TO PRINT
      TWO IS USED TO SIGNAL WHEN NOT TO PRINT A SECOND ARRAY
```

```
SUBROUTINE PROCES (PNODE,POMP)
*******************************************************
*  SUBROUTINE PROCESS PERFORMS RESPONSE ACTIONS BASED ON
*  CONDITIONS OF PROCESS RULES HAVING BEEN MET
*******************************************************
*  INPUT
*    PNODE  - POINTER TO NODE
*    POMP   - POINTER TO OUTPUT TYPE
*******************************************************
INCLUDE COMMONS
TEST FOR RANDOM MESSAGE PROCESSES
   CASE(OUTPUT MESSAGE TYPE)
*TYPES(2900,3000,3400,7000,9990,9993,3126,3130,3136,OTHER)
*CASE(REQUEST AIR SUPPORT)
   NON ALLOCATION PROCESS
   ALLOCATION PROCESS
*CASE(APPROVED AIR SUPPORT)
*CASE(DELETE ATO)
   ATO REQUEST FLAG
*CASE(REQUEST HELOCOPTER SUPPORT)
*CASE(APPROVED HELOCOPTER SUPPORT)
*CASE(DELETE HTO)
*CASE(ACCEPT STATUS REPORT FROM SUBORDINATE)
*CASE(RECEIVE SPOT INTEL REPORT ON RED FOE)
*CASE(CREATE MESSAGES FOR OUTPUT)
 CASE(RANDOM OUTPUT MESSAGE PROCESS)
*GET NEXT OUTPUT MESSAGE
*DELETE ALL ATO'S USED BY THIS PROCESS
```

```
      SUBROUTINE PRTATO(TITLE,PATO)
C**********************************************************
C* PRINTS AN AIR TASKING ORDER
C**********************************************************
      INCLUDE COMMONS
```

```
SUBROUTINE RANDIN
********************************************************
*  INPUT RANDOM DISTRIBUTIONS FROM FILE RANDIS AND
*  CREATE RAND STRUCTURE WITH BASE POINTER PRAND
********************************************************
INCLUDE COMMONS
READ IN HEADER DOCUMENTATION
READ PARTIAL AND CONTENT CHANGE MESSAGE DISTRIBUTIONS
INPUT IS COMPLETE
```

```
SUBROUTINE RANDOM(NDIST,VALUE)
**************************************************
*  RANDOM LOCATES THE DISTRIBUTION WITH INDEX NUMBER NDIST
*  AND RETURNS THE RANDOM VALUE BASED ON INTERPOLATION
*  BETWEEN THE TWO REFERENCED INDEXES FROM THE 11
*  DISTRIBUTION VALUES
**************************************************
*  INPUT
*     NDIST - INDEX OF DISTRIBUTION TO BE USED
**************************************************
*  OUTPUT
*     VALUE - INTERPOLATED RANDOM VALUE FROM DISTRIBUTION
**************************************************
INCLUDE COMMONS
LOCATE DESIRED DISTRIBUTION
   DISTRIBUTION TYPE NOT FOUND
DISTRIBUTION LOCATED
```

```
SUBROUTINE RANMSG(PNODE,POMP)
GET COMMONS
*****************************************************************
*  RANMSG CREATES ALL THE MESSAGES FOR A NODE THAT ARE CLASSIFIE
*  AS RANDOM. (IE PROCESS NUMBERS 3800, 4800, 5800, 6800, 7800,
*  5900, AND 7900.
*****************************************************************
INCLUDE COMMONS
TEST FOR FIRST TIME TO INITIALIZE NODE'S RANDOM QUEUE
      GET NEXT POUT

TEST FOR TIME TO SEND RANDOM MESSAGES
      SET UP GENERIC MESSAGE FOR NEXT RANDOM TIME
   GET NEXT RANDOM MESSAGE
```

```
SUBROUTINE RDRULE
*******************************************************
* THIS ROUTINE READS THE THREE SETS OF DATA THAT MAKE UP
* THE COMMAND POST RULES.
*******************************************************
READ IN THE PREAMBLE DOCUMENTATION
READ THE RULE DATA SET
   READ RULES
READ THE INPUT MESSAGE REQUIREMENTS
   READ INPUT MESSAGES
READ THE OUTPUT MESSAGES
   READ OUTPUT MESSAGES
END OF RULE INPUT
```

```
SUBROUTINE RELEAS (NPTR,LEN,ISPACE)
*************************************************************
*  SEGMENT RELEASE PUTS STORAGE ON GARBAGE LIST
*************************************************************
INCLUDE COMMONS
*IOUT IS OUTPUT DEVICE
*CHECK BAD PTR, LEN
**DO UNTIL NO GARBAGE EQUAL LENGTH
*END DO
*SNAP IN SPACE
*GARBAGE LENGTH NOT KNOWN
*PUT STORAGE ON GARBAGE LIST
*END SEGMENT
```

```
SUBROUTINE REPORT
****************************************************************
* THIS ROUTINE OUTPUTS THE INPUTS AND SELECTED COMPUTED VALUES I
* A FORMAT
****************************************************************
INCLUDE COMMONS
OUTPUT THE TITLE
OUTPUT THE INDEX WEAPON
OUTPUT THE BLUE VALUES
OUTPUT THE RED VALUES
OUTPUT K (BLUE AND RED)
```

```
SUBROUTINE RESEND(PMSG,PNODE)
*****************************************************
*  RESEND RESPONDS TO A REQUEST TO RESEND A MESSAGE
*  THE DATA TO BE RESENT IS IN THE MESSAGE'S DATA STRUCTURE
*  THERE IS NO PROCESS DELAY. THE OUTPUT MESSAGE IS PUT
*    ON THE FUTURE QUEUE
*****************************************************
*  INPUT
*    PMSG - POINTER TO MESSAGE
*    PNODE - POINTER TO PROCESSING NODE
*****************************************************
INCLUDE COMMONS
   ERROR CONDITION, NO DATA
CREATE RESEND MESSAGE
```

```
SUBROUTINE RESTOR
*********************************************************
* DYNAMIC MEMORY AND COMMON VALUES ARE READ FROM A FILE
* INTO MEMORY AS PHYSICAL STRUCTURES. THIS FILE IS
* CREATED BY SUBROUTINE STORE.
*********************************************************
INCLUDE COMMONS
RESTORE COMMON VARIABLES
RESTORE DYNAMIC MEMORY
```

```
SUBROUTINE RPTLOS(PNODE,BL,RL)
*****************************************************
* RPTLOS CREATES SPOT LOSS REPORTS FOR INTERNAL SYSTEMS
*****************************************************
* INPUT
*   PNODE  - POINTER TO NODE STRUCTURE
*   BL     - ARRAY OF BLUE WEAPON SYSTEM LOSSES
*   RL     - ARRAY OF RED  WEAPON SYSTEM LOSSES
*****************************************************
INCLUDE COMMONS
INITIALIZE SPOT REPORT
COPY LOSSES
COPY ENGAGEMENT RATE
PUT MESSAGES ON THE FUTURE QUEUE IN TIME ORDER
```

```
SUBROUTINE RULEIN
*******************************************************
* CREATES OUTPUT MESSAGE PROCEDURES AND REQUIRED INPUT
* MESSAGE TYPES FOR EACH NODE. ALSO CREATES THE GENERIC
* OUTPUT MESSAGES FOR EACH RULE. GETS DATA FROM
* COMMON/RULE/ SET BY BLOCK DATA RULES.
*******************************************************
INCLUDE COMMONS
INCLUDE READ RULE DATA
DO FOR EACH PROCESS RULE IN BLOCK DATA
CREATE OUTPUT MESSAGE QUEUE
   CHECK EXISTING QUEUES FOR THIS RULE NUMBER
   NEW RULE
   RULE EXISTS, SET POINTER & SKIP CREATE QUEUE
   FIND NEXT MESSAGE FOR THIS RULE
      PUT MESSAGE ON QUEUE
      GET MESSAGE STRUCTURE
      GENERIC MESSAGE COMPLETE, GET NEXT MESSAGE
   OUTPUT MESSAGE QUEUE COMPLETE
   DO FOR EACH NODE OF THIS TYPE
         GET RULE STRUCTURE
         DO FOR EACH MESSAGE REQUIRED FOR THIS RULE
            FIND NEXT INPUT MESSAGE FOR THIS RULE
            CREATE INPUT MESSAGE STRUCTURE
            GET AN INPUT MESSAGE STRUCTURE
            INITIALIZE INPUT MESSAGE QUEUE
         END OF INPUT MESSAGE PROCESSING
      GET NEXT NODE
   END OF NODES
   GET NEXT RULE
END OF RULES, PRINT RULES OUT
```

```
SUBROUTINE RULOUT
*******************************************************
* PRINT OUT RULES
*******************************************************
INCLUDE COMMONS
IOUT IS OUTPUT DEVICE
DO FOR ALL NODES
   DO FOR ALL INPUT MESSAGE TYPES
      DO FOR ALL INPUT MESSAGE LISTS
      NEXT INPUT MESSAGE TYPE
   PRINT OUT GENERIC MESSAGES
      GET NEXT MESSAGE
   GET NEXT POMP
GET NEXT NODE
LAST NODE
```

```
SUBROUTINE RULPRT
************************************************************
* ECHOES OUT THE INPUT RULES
************************************************************
 INCLUDE COMMONS
DO FOR ALL LEVELS
   DO FOR ALL PROCESSES
         DO FOR ALL INPUT MESSAGES
```

```
SUBROUTINE SAVE
********************************************************
* DYNAMIC MEMORY AND COMMON VALUES ARE WRITTEN TO A FILE
* AS PHYSICAL STRUCTURES. THIS FILE MAY BE USED TO
* RESTART THE SIMULATION AT THE POINT WHERE IT LEFT OFF.
********************************************************
INCLUDE COMMONS
SAVE COMMON VARIABLES
SAVE DYNAMIC MEMORY
```

```
SUBROUTINE SEND
****************************************************
*  MOVES MESSAGES FROM ORIGINATOR SEND QUEUE TO
*  DESTINATION INPUT QUEUE, UPDATES LINK CAPACITIES AND
*  DELETES OUT OF TIME MESSAGES ON HOLD QUEUES
*  MAY DELETE MESSAGES RANDOMLY
****************************************************
*  CALLS - FIND, RELEAS
****************************************************
INCLUDE COMMONS
DO FOR ALL NODES
   DO FOR ALL DESTINATIONS
      GET DESTINATION NODE
      DO FOR ALL LINKS
         INCREMENT MESSAGE SENT COUNTER
        TEST FOR RANDOM PROCESSING
           GET DISTRIBUTION TYPE
              DELETE MESSAGE (LOST)
                 DELETE MESSAGE DATA
         MOVE CONTENTS OF SEND QUEUE TO DESTINATION INPUT
         MERGE CONTENTS OF SEND AND INPUT QUEUES
         FIND LAST MESSAGE ON SEND QUEUE
         LAST MESSAGE FOUND
         GET NEXT LINK
         DO FOR ALL MESSAGES ON HOLD
            CHECK AGE OF MESSAGE
               DELETE THIS MESSAGE
                  DELETE DATA STRUCTURE
               DELETE MESSAGE STRUCTURE
         GET NEXT MESSAGE
      GET NEXT LINK
   GET NEXT DESTINATION
GET NEXT NODE
LAST NODE
```

```
      SUBROUTINE SNAP(PTR, NWORD, PIN,ISPACE)
*************************************************************
*  SUBROUTINE SNAP PLACES A RECORD IN SORTED ORDER IN A QUEUE
*************************************************************
*  INPUT
*    PTR    - POINTER TO ROOT OF QUEUE
*    NWORD  - OFFSET IN RECORD STRUCTURE FOR SORT VALUE
*    PIN    - POINTER TO RECORD TO BE INSERTED
*    ISPACE - ARRAY THAT CONTAINS DYNAMIC MEMORY
*************************************************************
*EMPTY QUEUE - MAKE FIRST RECORD
*INSERT BEFORE RECORD
*INSERT BEFORE 1ST RECORD
*INSERT AFTER LAST RECORD
```

```
SUBROUTINE STATIN
******************************************************
*  STATIN PERFORMS INITIATION ACTIONS FOR THE COMMANDERS
*  PERCEPTION OF SUBORDINATE STRENGTHS AND COMBAT STATUS
******************************************************
INCLUDE COMMONS
DO FOR ALL NODES
   DO FOR ALL STAT (SUBORDINATES) BLOCKS
      SET BLUE STRENGTHS
```

```
SUBROUTINE STATOU
*****************************************************
*  STATOU PRINTS OUT THE COMMANDERS
*  PERCEPTION OF SUBORDINATE STRENGTHS AND COMBAT STATUS
*****************************************************
INCLUDE COMMONS
DO FOR ALL NODES
   DO FOR ALL STAT (SUBORDINATES) BLOCKS
      GET BLUE STRENGTHS
      GET ENGAGEMENT RATES
```

```
SUBROUTINE STATUP (PNODE,POMP)
*******************************************************
*  SUBROUTINE STATUP UPDATES COMMANDERS PERSEPTIONS OF
*  SUBORDINATES COMBAT STATUS VIA MESSAGES 3126 AND 3130
*******************************************************
*  INPUT
*    PNODE  - POINTER TO NODE
*    POMP   - POINTER TO OUTPUT TYPE
*******************************************************
INCLUDE COMMONS
TEST FOR SUBORDINATE STATUS STRUCTURE
GET FIRST SPOT REPORT
IF REPORT MATCHES PROCESS, LOG DATA
  FIND SUBORDINATE'S STATUS STRUCTURE
     ERROR, NO SUBORDINATE STATUS STRUCTURE
  DETERMINE SIDE
     UPDATE BLUE LOSSES AND STRENGTHS
     UPDATE BLUE ENGAGEMENT RATE
     UPDATA BLUE DATA TIME TO LATEST TIME
     UPDATE RED LOSSES
     UPDATE RED ENGAGEMENT RATE
     UPDATA RED DATA TIME TO LATEST TIME
  GET NEXT SPOT REPORT
    ALL SPOT REPORTS PROCESSED FOR THIS RULE
```

```
SUBROUTINE STATUS(MODE,TY1S, ICOUNT, PREADY)
*****************************************************
* PRINT WOC AIRCRAFT STATUS
*****************************************************
INCLUDE COMMONS
```

```
SUBROUTINE SUPOPS
**************************************************************
*  RETURNS HELICOPTERS TO COMMANDER EACH TIME CYCLE
*  WITHDRAWS ARTILLERY SUPPORT
*  UPDATES HELICOPTER AVAILABILITY FOR ASSIGNMENT
*  ALLOCATES CAS, HELICOPTERS AND ARTILLERY TO SUBORDINATES
*  UPDATES SUBORDINATES COMBAT SUPPORT ARTILLERY AND
*    HELICOPTER SYSTEMS IN COMBAT MATRICIES
**************************************************************
INCLUDE COMMONS
RETURN COMBAT SUPPORT
   COMMANDER NODE MUST ALLOCATE
   COMMANDER MUST HAVE 3400 ALLOCATION PARAMETERS
      WITHDRAW HELICOPTER AND ARTILLERY SUPPORT
   GET NEXT NODE
UPDATE COMBAT SUPPORT SYSTEMS AVAILABILITY AND ALLOCATE
   GET AVAILABILITY
   PROCESS NODES WITH COMBAT SUPPORT SYSTEMS
   GET NEXT NODE
UPDATE COMBAT UNITS' COMBAT SUPPORT SYSTEMS
   CHECK FOR COMBAT SUPPORT TYPE UNIT
          CHECK FOR HELICOPTER SUPPORT
             MOVE ALL ARRIVING HELICOPTERS TO COMBAT MATRIX
            CHECK TIME ON TARGET
                SET HELICOPTER SUPPORT
                INCREMENT HELICOPTER SORTIE COUNT
                RESET UNIT SUPPORT REQUEST TO NULL
         GET NEXT ATO
      PRINT HELICOPTER TAKEOFF
      GET NEXT ATO
      PUT ARTILLERY TUBES INTO COMBAT MATRIX
      NOT IMPLEMENTED YET
   GET NEXT NODE
END SUPOPS
```

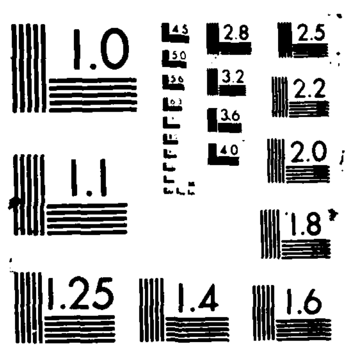
```
SUBROUTINE TABLES(SIDEO,SIDED,NAMEX,NAMEY,NTX,NTY,PARAM)
*******************************************************
*  THIS ROUTINE IS USED TO OUTPUT 2 DIMENSIONAL ARRAYS
*******************************************************
INCLUDE COMMONS
LUPLIM IS THE NUMBER OF COLUMNS TO PRINT
```

```
SUBROUTINE TIMOUT(HEAD)
*****************************************************
* PRINT OUT EACH MESSAGE THAT HAS ITS OUTPUT FLAG SET
* OUTPUT FILE IS FOR025.DAT
*****************************************************
INCLUDE COMMONS
NOUT IS DEBUG FILE
DO FOR ALL NODES
   DO FOR ALL DESTINATIONS
      DO FOR ALL LINKS
         DO FOR ALL MESSAGES ON HOLD
         GET NEXT MESSAGE
      REPEAT FOR SEND QUEUE
      GET NEXT LINK
   GET NEXT DESTINATION
GET NEXT NODE
LAST NODE
```

```
SUBROUTINE TINPUT
*****************************************************
* INPUT CHANGES TO CHARACTERISTICS DURING TIME T
* USES UNFORMATTED TEMPORARY FILES 11, 12, 13
*****************************************************
INCLUDE COMMONS
   GET FIRST MESSAGES
CHECK FOR CURRENT REINFORCEMENTS
CHECK FOR CURRENT NODE CHAGES
CHECK FOR CURRENT LIMIT CHANGES
CHECK FOR CURRENT LINK CHANGES
CHECK FOR CURRENT ALLOCATION PARAMETER CHANGES
   FIND NODE
   FIND ALLOCATION STRUCTURE FOR TYPE MSG
END TIME T INPUT FOR THIS TIME INCREMENT
```

```
SUBROUTINE VMDATA (LEN)
INCLUDE COMMONS
    *DO UNTIL LIST ENDS
```

AD-A186 370 C3EVAL MODEL DEVELOPMENT--1986 VOLUME 2 PROGRAMMERS' 4/4
MANUAL(U) INSTITUTE FOR DEFENSE ANALYSES ALEXANDRIA VA
R F ROBINSON ET AL. APR 87 IDA-P-1978-VOL-2
UNCLASSIFIED IDA/HQ-86-31573 MDA903-84-C-0031 F/G 12/5 NL


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

IV. PROGRAM POSTPROC

The post processor consists of two options. Both options produce graphics from data files created by C3EVAL. Option GRAPHSUM uses file C3SUM.DAT for input to create summary graphs. Option GRAPHT uses files TIMET.DAT and LOSST.DAT for input to create TIMET graphs. File C3SUM contains all summary data pertaining to the number of messages, number of sorties and number of combat system losses. File TIMET contains running totals for each time period for all data pertaining to the number of messages and sorties. File LOSST contains running totals for each time period for all data pertaining to the number of combat system losses.

A. SUBROUTINE GRAPHSUM

GRAPHSUM creates bar graphs using the summary data output by C3EVAL. The actual graphing will be done by a call to VAXDECGRAPH. Any detailed information on DECGRAPH can be obtained from the appropriate VAXDECGRAPH manuals. There are five types of graphs:

- Communications path limits,
- Input message limits,
- Output message limits,
- Combat support and
- Losses.

The graphs represent the total messages/sorties/losses for a given time period for each unit displayed. The first four graphs can display from one to five units at a time; the fifth graph can only display one unit at a time.

1. Subroutine GETSUM

The first part of file C3SUM is the preamble documentation. The first line of the preamble documentation is used as the subtitle for the graph. The last line contains "END" in card columns 1-4. Note that the preamble must contain at least two lines. After the "END" card, three header documentation lines follow. The rest of the file is data. There is a maximum of 40 units, each represented by one data line of the form (1X,3A4,2I8,12I5). The values read are:

Unit identifier (consisting of 12 characters),
Unit number,
Unit type,

Communications limit in,
Communications limit out,
Communications limit held,
Communications limit deleted,
Input limit in,
Input limit held,
Input limit deleted,
Output limit out,
Output limit held,
Output limit deleted,
Preplanned and immediate CAS sorties and
Non-division helicopter sorties.

The last unit read in must be the unit number one.

The second data part of file C3SUM is the cumulative losses, flagged by a row of asterisks (*). This section contains all units that sustained any losses since time 0. Each data line is in free format, containing time, unit number, number of Blue losses for each combat system for the specified unit and number of red of Red losses for each combat system for the specified unit.

2. Subroutine GETSYS2

This subroutine is only called when the user requests to graph losses. The user is allowed to choose from one to six combat systems to graph from a list of 11 combat systems. The 11 systems are:

- APC,
- AFV,
- Tank,
- Atank lt,
- Atank hv,
- Mortar,
- Artillery,
- Helicopter,
- AAA,
- SAM and
- CAS.

The user then chooses which unit to graph. The losses for the specified Blue unit and the Red units facing the Blue unit are graphed. The units that the user can choose from are all units that have non-zero losses.

3. Subroutine GETVECTOR

This subroutine determines which units will be graphed. There is a limit of five units per graph. The units to be graphed are determined at run time by the user. When the user selects the units, their locations within the data structure are stored in a unit vector for later recall. The units available for selection are determined by the input file.

4. Subroutine GRAPH6

Subroutine GRAPH6 creates the data file for a bar graph of the communications path limits. The data file consists of two sections. The instruction portion contains the title, subtitle, horizontal label and vertical label for the graph. The data portion has a line for each unit in the unit vector. Each line contains the unit name and the number of messages in, out, held and deleted by the communications link.

5. Subroutine GRAPH7

Subroutine GRAPH7 creates the data file for a bar graph of the input message limits. The data file consists of two sections. The instruction portion contains the title, subtitle, horizontal label and vertical label for the graph. The data portion has a line for each unit in the unit vector. Each line contains the unit name and number of messages in, held and deleted at the input side of the node.

6. Subroutine GRAPH8

Subroutine GRAPH8 creates the data file for a bar graph of the output message limits. The data file consists of two sections. The instruction portion contains the title, subtitle, horizontal label and vertical label for the graph. The data portion has a line for each unit in the unit vector. Each line contains the unit name and the number of messages out, held and deleted at the output side of the node.

7. Subroutine GRAPH9

Subroutine GRAPH9 creates the data file for a bar graph of the combat support. The data file consists of two sections. The instruction portion contains the title, subtitle, horizontal label and vertical label for the graph. The data portion has a line for each unit in the unit vector. Each line

contains the unit name and the number of close air support and non-division helicopter sorties the unit received.

8. Subroutine GRAPH10

Subroutine GRAPH 10 creates the data file for a bar graph of losses. The data file consists of two sections. The instruction portion contains the title, subtitle, horizontal label and vertical label for the graph. The title contains the name of the unit being graphed. The data portion has a line for each combat system to be graphed. Each line contains the name of the combat system and the Blue and Red losses at the specified unit.

B. SUBROUTINE GRAPHT

GRAPHT creates line graphs using the data output by C3EVAL. The actual graphing will be done by a call to VAXDECGRAPH. Any detailed information on DECGRAPH can be obtained from the appropriate VAXDECGRAPH manuals. There are five types of graphs:

- Communications path limits,
- Input message limits,
- Output message limits,
- Combat support, and
- Losses.

The graphs represent the number of messages/sorties/losses for a given unit for each time increment over a set time period.

1. Subroutine GETTIMET

The input file TIMET consists of a multiple number of data sets. Each data set represents the running totals for a given time period. Therefore, in order to obtain the number of messages/sorties for each time increment, the previous total is subtracted from the current total. The first part of the file is the preamble documentation. The first line of the preamble documentation is used as the subtitle for the graph. The last line of the preamble contains "END" in card columns 1-4. Note: The data sets follow the "END" card.

Each data set starts with a line of the form (23X,I6) where the value is the time corresponding to the data. The second and third lines are header documentation lines. The rest of

the data set is data. There is a maximum of 40 units each represented by one data line of the form (1X,3A4,2I8,12I5). The values read are:

Unit identifier (consisting of 12 characters),
Unit number,
Unit type,
Communications limit in,
Communications limit out,
Communications limit deleted,
Input limit in,
Input limit held,
Input limit deleted,
Output limit out,
Output limit held,
Output limit deleted,
Preplanned and immediate CAS sorties and
Non-division helicopter sorties.

The last line read in for each data set must be the unit number one.

The input file LOSST is in similar format to the input file TIMET, except that it contains the information on combat losses instead of information on the number of messages and sorties. The other differecet is that LOSST is in binary format instead of ASCII format. Each data line consists of time, unit number, number of Blue losses for each combat system for the specified unit and the number of Red losses for each combat system for the specified unit.

2. Subroutine GETSYS1

This subroutine is only called when the user requests to graph losses. The user is allowed to choose from one to six combat systems to graph from a list of 11 combat systems. The 11 combat systems are:

- APC,
- AFV,
- Tank,
- Atank lt,
- Atank hv,
- Mortar,
- Artillery,
- Helicopter,
- AAA,
- SAM and
- CAS.

Then the user selects either the Red side, Blue side or Red and Blue together to be graphed and which particular unit to graph. The units that the user can choose from are all units that have non-zero losses.

3. Subroutine GETUNIT

This subroutine determines which unit will be graphed. The unit to be graphed is determined at run time by the user. When the user selects the unit, its location within the data structure is stored for later recall. The units available for selection are determined by the input file.

4. Subroutine GRAPH1

Subroutine GRAPH1 creates the data file for a line graph of the communications path limits at a specified node over a given time interval. The data file consists of the instruction portion and the data portion. The instruction portion contains the title, subtitle, horizontal label and vertical label for the graph. The data portion has a line for each time increment. Each line contains the time and the number of messages in, out, held and deleted by the communications links for that one time increment.

5. Subroutine GRAPH2

Subroutine GRAPH2 creates the data file for a line graph of the input message limits at a specific node over a given time interval. The data file consists of two sections. The instruction portion contains the title, subtitle, horizontal label and vertical label for the graph. The data portion has a line for each time increment. Each line contains the time and the number of messages in, held and deleted at the input side of the node for that one time increment.

6. Subroutine GRAPH3

Subroutine GRAPH3 creates the data file for a line graph of the output message limits at a specific node over a given time interval. The data file consists of two sections. The instruction portion contains the title, subtitle, horizontal label and vertical label for the graph. The data portion has a line for each time increment. Each line contains the time and the number of messages out, held and deleted at the input side of the node for that one time increment.

7. Subroutine GRAPH4

Subroutine GRAPH4 creates the data file for a line graph of the combat support at a specific node over a given time interval. The data file consists of two sections. The instruction portion contains the title, subtitle, horizontal label and vertical label for the graph. The data portion has a line for each time increment. Each line contains the time and the number of close air support and helicopter sorties the node received for that one time increment.

8. Subroutine GRAPH5

Subroutine GRAPH5 creates the data file for a line graph of the losses at a specific node for either the Red side, Blue side or both over a given time interval.. The data file consists of two sections. The instruction portion contains the title, subtitle, horizontal label and vertical label for the graph. The title contains the name of the unit to be graphed and which side is to be graphed. The data portion has a line for each time increment. Each line contains the time and the number of losses for each combat system for the specified side at the specified node.

C. UTILITIES

1. Subroutine CHRINT

Same as Subroutine CHRINT in Chapter II, Program PREPROC, Section N.1.

2. Subroutine DIS_SYS

Subroutine DIS_SYS displays the combat systems along with their flags in the scrolled area. Initializes the variables which are used to facilitate the scrolling functions to allow viewing of the combat systems which are not currently displayed in the scrolled area.

3. Subroutine DIS_UNITS

Subroutine DIS_UNITS is similar to Subroutine DIS_SYS except it displays the unit numbers along with their corresponding unit name and flag instead of the combat systems.

4. Subroutine DIS_UNITS2

Subroutine DIS-UNITS2 is the same as Subroutine DIS_UNITS except it only displays those units that sustained combat losses.

5. Subroutine INTCHR

Same as Subroutine INTCHR in Chapter II, Program PREPROC, Section N.8.

6. Subroutine OPTIONS

Allows the user to choose what action to take and also allows him to turn the printing of the graphs on and off . The options to choose from are:

1) Quit,
2) Load a new file,
3) Graph communications path limits,
4) Graph input limits,
5) Graph Output limits,
6) Graph combat support and
7) Graph combat losses.

7. Subroutine SCRBK

Performs same function as Subroutine SCRBK in Chapter II, Program PREPROC, Section N.15.

8. Subroutine SCRBK2

Same as Subroutine SCRBK, except it only displays those units that sustained combat losses.

9. Subroutine SCRFWD

Performs same function as Subroutine SCRFWD in Chapter II, Program PREPROC, Section N.16.

10. Subroutine SCRFWD2

Same as Subroutine SCRFWD, except only displays those units that sustained combat losses.

11. Subroutine SCRLINE

Subroutine SCRLINE is used to create the line to output to the scrolled area for units. Each line contains three fields consisting of:

- Unit number,
- Unit name and
- Graphics flag.

The flag identifies whether or not to graph that particular combat system.

12. Subroutine SCRLINE2

Subroutine SCRLINE2 is used to create the line to output to the scrolled area for combat systems. Each line contains two fields consisting of:

- Combat system name and
- Flag.

The flag identifies whether or not to graph that particular combat system.

13. Subroutine SCRLINE3

Same as Subroutine SCRLINE, except only displays those units who that sustained combat losses.

14. Subroutine VALID1

Performs same function as Subroutine VALID1 in Chapter II, Program PREPROC, Section N.20.

D. DATA STRUCTURES

There are two major functions within program POSTPROC. These two functions are TIMET line graphs and summary bar graphs. The data structures for the TIMET graphs allow for

multiple time values, whereas the data structures for the summary graphs allow for only one time value.

1. Module Summary Graphics

NAME	TYPE	DESCRIPTION
COMSYS	A	1st dimension - Multiple combat systems 2nd dimension 1-3 - Name of the combat system 4 - Combat system's number
LOSS	A	1st dimension - Multiple units 2nd dimension 1-11 - Blue combat system losses 12-22 - Red combat system losses 23 - Force ratio
NUMELEM	I	Number of units to graph
NUMSYS	I	Number of combat systems to graph
NUMUNIT	I	Number of units read in from data file
SCREEN	I	Value representing the action to take: 1 - Quit 2 - Communications path limits 3 - Input limits 4 - Output limits 5 - Combat support 6 - Losses
SUBTI	A	40-character field for identifying the data set
SUMDATA	A	1st dimension - Multiple units 2nd dimension 1-4 - Communication paths limits Number of messages in, out, held and deleted 5-7 - Input limits Number of messages in, held and deleted 8-10 - Output limits Number of messages out, held and deleted 11-12 - Combat support Number of CAS and helicopter sorties

UNITID	A	1st dimension - Multiple units 2nd dimension 1-3 - Unit name 4 - Unit number 5 - Unit type 6 - Combat system losses flag 0 --> unit had no losses 1 --> unit suffered losses
VECTOR	A	Unit numbers of the units to graph

2. Module TimeT Graphics

NAME	TYPE	DESCRIPTION
COMSYS	A	1st dimension - Multiple combat systems 2nd dimension 1-3 - Name of the combat system 4 - Combat system's number
LOSS	A	1st dimension - Multiple time values 2nd dimension - multiple units 3rd dimension 1-11 - Blue combat systems losses 12-22 - Red combat systems losses 23 - Force ratio
NUMSYS	I	Number of combat systems to graph
NUMUNIT	I	Number of units read in from data file
SCREEN	I	Value representing the action to take: 1 - Quit 2 - Communications path limits 3 - Input limits 4 - Output limits 5 - Combat support 6 -Losses
SIDE	I	Value representing which side to graph: 1 - Blue 2 - Red
SUBTI	A	40-character field for identifying the data set

SUMDATA	A	1st dimension - Multiple time values 2nd dimension - Multiple units 3rd dimension 1-4 - Communication paths limits Number of messages in, held and deleted 5-7 - Input limits Number of messages out, held and deleted 8-10 - Output limits Number of messages in, out, held and deleted 11-12 - Combat support Number of CAS and helicopter sorties
TIME	I	Number of time increments read in
TIMET	A	Value of each time increment read in
UNIT	I	Unit number of the unit to graph
UNITID	A	1st dimension - Multiple units 2nd dimension 1-3 - Unit name 4 - Unit number 5 - Unit type 6 - Combat system losses flag 1 --> unit suffered losses 0 --> unit had no losses

E. PROGRAM NOTES

One possible enhancement would be to add the capability for the user to specify the range for the vertical axis. In this way, the graphs would have the same scales and could be placed side to side for comparison. The problem is that there does not seem to be any way to tell DECGRAPH to freeze the scale from the user's code. Another enhancement would be to specify colors for each part of the graph, instead of leaving it to random selection.

F. INTERNAL CODE DOCUMENTATION

PROGRAM POSTPROC

INITIALIZE FMS

FMS CLEAN UP

SUBROUTINE CHRINT(STRING, ISIZE, VALUE)

SUBROUTINE CONVERTS CHARACTER STRING REPRESENTATION OF
AN INTEGER INTO IT'S INTEGER VALUE

PARAMETERS:

```
   STRING    ==>   STRING TO CONVERT TO INTEGER EQUIVALENT
   ISIZE     ==>   NUMBER OF CHARACTERS TO CONVERT
   VALUE     ==>   INTEGER RESULT OF CONVERSION
```

```
SUBROUTINE DIS_SYS(NAME,FLAG,MAXSYS,MAXLINE,LINE,CURPOS)

PURPOSE:  DISPLAY THE COMBAT SYSTEMS ALONG WITH THEIR FLAGS
  IN THE SCROLLED AREA.

PARAMETERS:
  NAME      => ARRAY OF NAMES OF COMBAT SYSTEMS
  FLAG      => ARRAY OF FLAGS FOR COMBAT SYSTEMS
  MAXSYS    => NUMBER OF COMBAT SYSTEMS
  MAXLINE   => NUMBER OF LINES IN SCROLLED AREA
  LINE      => LINE NUMBER OF THE CURRENT LINE OF THE
               SCROLLED AREA
  CURPOS    => INDICE INTO COMBAT SYSTEM ARRAYS. POINTS
               TO THE CURRENT ELEMENT.

FMS TERMINATOR CODES

DISPLAY AS MANY COMBAT SYSTEMS AS POSSIBLE IN THE SCROLLED ARE

IF NUMBER OF COMBAT SYSTEMS IS LESS THAN NUMBER OF LINES IN
SCROLLED AREA THEN DISPLAY APPROPRIATE NUMBER OF BLANK LINES
```

SUBFOUTINE DIS_UNITS(MAXARRAY,MAXLINE,LINE,CURPOS)

PURPOSE: DISPLAY UNIT NUMBERS ALONG WITH THEIR CORRESPONDING NAMES AND FLAGS IN THE SCROLLED AREA.

PARAMETERS:

MAXARRAY	=>	NUMBER OF UNITS IN NODE QUEUE
MAXLINE	=>	NUMBER OF LINES IN SCROLLED AREA
LINE	=>	LINE NUMBER OF THE CURRENT LINE OF THE SCROLLED AREA
CURPOS	=>	INDICE INTO NODE QUEUE. POINTS TO THE CURRENT ELEMENT.

FMS TERMINATOR CODES

SUBROUTINE DIS_UNITS2(MAXARRAY,MAXLINE,LINE,CURPOS)

PURPOSE: DISPLAY UNIT NUMBERS ALONG WITH THEIR CORRESPONDING NAMES AND FLAGS IN THE SCROLLED AREA. NOTE: ONLY THOSE UNITS WHO HAD LOSSES ARE DISPLAYED.

PARAMETERS:

```
  MAXARRAY => NUMBER OF UNITS IN NODE QUEUE
  MAXLINE  => NUMBER OF LINES IN SCROLLED AREA
  LINE     => LINE NUMBER OF THE CURRENT LINE OF THE
              SCROLLED AREA
  CURPOS   => INDICE INTO NODE QUEUE. POINTS
              TO THE CURRENT ELEMENT.
```

FMS TERMINATOR CODES

```
SUBROUTINE INTCHR(VALUE, ISIZE, STRING)

PURPOSE:  CONVERTS AN INTEGER TO ITS ASCII REPRESENTATION

PARAMETERS:
   VALUE    --> INTEGER VALUE TO CONVERT
   LENGTH   --> NUMBER OF DIGITS TO CONVERT
   STRING   --> ASCII REPRESENTATION OF VALUE
```

```
SUBROUTINE GETSUM(LOSS,UNITID,SUBTI,SUMDATA,NUMUNIT,B,R,

PURPOSE:  TO READ SUMMARY NODE MESSAGE DATA FILE AND SUMMARY
  COMBAT LOSSES DATA FILE.

PARAMETERS:
  LOSS     => ARRAY CONTAINING COMBAT LOSSES FOR EACH UNIT
  UNITID   => ARRAY CONTAINING UNIT NAMES AND NUMBERS
  SUBTI    => SUBTITLE TO PRINT WITH GRAPH
  SUMDATA  => ARRAY CONTAINING MESSAGE DATA FOR EACH UNIT
  NUMUNIT  => NUMBER OF UNITS TO CHOOSE FROM
  B        => ARRAY OF BLUE LOSSES FOR ONE UNIT FOR ONE TIME
              INCREMENT
  R        => ARRAY OF RED LOSSES FOR ONE UNIT FOR ONE TIME
              INCREMENT
  RAT4     => BLUE ON RED FORCE RATIO
  FILE1    => FILE TO READ NODE MESSAGE DATA FROM
  MAXSYS   => TOTAL NUMBER OF COMBAT SYSTEMS TO CHOOSE FROM
  ITIME    => THE TIME CORRESPONDING TO THE SUMMARY DATA

FMS TERMINATOR CODES

CONSTANT

ALLOW USER TO CHANGE DEFAULT FILE NAMES TO LOAD FROM

PROCESS FIELD TERMINATOR
OPEN FILES

INITIALIZE DOUBLE DIMENSIONED ARRAYS TO ALL ZERO'S

READ SUBTITLE

SKIP OVER THE PREAMBLE DOCUMENTATION

SKIP OVER THE HEADER DOCUMENTATION

READ IN THE NODE MESSAGE DATA

READ UNTIL ENCOUNTER '*****'

READ LOSSES DATA
```

SUBROUTINE GETSYS2(VECTOR,NUMSYS,NUMUNIT,NAME,FLAG,MAXSYS)

PURPOSE:
USED WHEN GRAPHING LOSSES. DETERMINES WHICH COMBAT SYSTEM LOSSES TO GRAPH. EACH GRAPH IS LIMITED TO THE LOSSES AT A PARTICULAR NODE.

PARAMETERS:

```
VECTOR  -> ARRAY OF POINTERS TO WHICH UNITS TO GRAPH
NUMSYS  -> NUMBER OF COMBAT SYSTEMS TO GRAPH
NUMUNIT -> NUMBER OF UNITS TO CHOOSE FROM
NAME    -> ARRAY OF COMBAT SYSTEM NAMES
FLAG    -> ARRAY OF FLAGS CORRESPONDING TO ARRAY NAME.
           0  ->  COMBAT SYSTEM HAS NOT BEEN CHOSEN FOR GRAP
           1  ->  COMBAT SYSTEM HAS BEEN CHOSEN FOR GRAPHING
MAXSYS  -> NUMBER OF COMBAT SYSTEMS TO CHOOSE FROM
```

FMS TERMINATOR CODES

DETERMINE WHICH COMBAT SYSTEMS TO GRAPH

PROCESS FIELD TERMINATOR

SAVE THE NAMES AND NUMBERS OF THE COMBAT SYSTEMS CHOSEN FOR GR

DETERMINE WHICH UNIT TO GRAPH. ONLY UNITS WHO HAVE NON-ZERO LOSSES ARE SELECTABLE FOR GRAPHING.

PROCESS FIELD TERMINATOR

SUBROUTINE GRAPHSUM(WORKSPACE,TCA,PRINT,FLAG,NUMSYS)

PURPOSE: CREATE BAR GRAPHS USING SUMMARY DATA OUTPUTTED BY C3EVAL. THERE ARE 5 TYPES OF GRAPHS: COMMUNICATIONS PATH LIMIT, INPUT LIMIT, OUTPUT LIMIT, COMBAT SUPPORT AND LOSSES

LIMITATIONS:
MAXIMUM OF 40 UNITS

EXTERNAL REFERENCES:
DECGRAPH

CONSTANT

CALL SUBROUTINE GETSUM TO LOAD IN MESSAGE DATA FILE AND COMBAT LOSSES FILE

INITIALIZE GRAPH DATA FILE

SELECT GRAPH

LOAD MESSAGE DATA FILE AND COMBAT LOSSES FILE

CREATE DATA FILE FOR GRAPH 1 - COMMUNICATIONS PATH LIMIT

CREATE DATA FILE FOR GRAPH 2 - INPUT LIMIT

CREATE DATA FILE FOR GRAPH 3 - OUTPUT LIMIT

CREATE DATA FILE FOR GRAPH 4 - COMBAT SUPPORT

CREATE DATA FILE FOR GRAPH 5 - LOSSES

CLOSE DATA FILE. CLOSE FMS STUFF.

C[illegible] GRAPH

W[illegible]T UNTIL USER READY TO CONTINUE

RE-INITIALIZE FMS STUFF

SUBROUTINE GRAPHT(WORKSPACE,TCA,PRINT,FLAG,NUMSYS)

PURPOSE: CREATE LINE GRAPHS USING TIME T DATA OUTPUTTED BY C3EVAL. THERE ARE 4 TYPES OF GRAPHS: COMMUNICATIONS PATH LIMIT, INPUT LIMIT, OUTPUT LIMIT, AND COMBAT SUPPORT.

LIMITATIONS:
THE LAST LINE OF EACH TIME T DATA SET MUST BE THE UNIT WHOSE IDENTIFIER IS 1.

EXTERNAL REFERENCES:
DECGRAPH

CONSTANTS

INITIALIZE GRAPH DATA FILE

SELECT GRAPH

LOAD MESSAGE DATA FILE AND COMBAT LOSSES FILE

CREATE DATA FILE FOR GRAPH 1 - COMMUNICATIONS PATH LIMIT

CREATE DATA FILE FOR GRAPH 2 - INPUT LIMIT

CREATE DATA FILE FOR GRAPH 3 - OUTPUT LIMIT

CREATE DATA FILE FOR GRAPH 4 - COMBAT SUPPORT

CREATE DATA FILE FOR GRAPH 5 - LOSSES

CLOSE DATA FILE. CLOSE FMS STUFF

CREATE GRAPH

WAIT UNTIL USER READY TO CONTINUE

RE-INITIALIZE FMS STUFF

```
SUBROUTINE GETSYS1(UNIT,NUMSYS,NUMUNIT,NAME,FLAG,MAXSYS,SIDE)

PURPOSE:
   USED WHEN GRAPHING LOSSES.  DETERMINES WHICH COMBAT SYSTEM
LOSSES TO GRAPH.  EACH GRAPH IS LIMITED TO THE LOSSES AT A
PARTICULAR NODE ON EITHER THE RED OR BLUE SIDE.

PARAMETERS:
  UNIT    => POINTER TO WHICH UNIT TO GRAPH
  NUMSYS  => NUMBER OF COMBAT SYSTEMS TO GRAPH
  NUMUNIT => NUMBER OF UNITS TO CHOOSE FROM
  NAME    => ARRAY OF COMBAT SYSTEM NAMES
  FLAG    => ARRAY OF FLAGS CORRESPONDING TO ARRAY NAME.
             0  ->   COMBAT SYSTEM HAS NOT BEEN CHOSEN FOR GRAP
             1  ->   COMBAT SYSTEM HAS BEEN CHOSEN FOR GRAPHING
  MAXSYS  => NUMBER OF COMBAT SYSTEMS TO CHOOSE FROM
  SIDE    => WHICH SIDE(S) TO GRAPH
             1  ->   GRAPH BLUE
             2  ->   GRAPH RED
             3  ->   GRAPH BLUE & RED


FMS TERMINATOR CODES


DETERMINE WHETHER TO GRAPH BLUE OR RED SIDE
  OR BOTH SIDES


DETERMINE WHICH COMBAT SYSTEMS TO GRAPH


PROCESS FIELD TERMINATOR


SAVE THE NAMES AND NUMBERS OF THE COMBAT SYSTEMS CHOSEN FOR GR


DETERMINE WHICH UNIT TO GRAPH. ONLY UNITS WHO HAVE NON-ZERO
LOSSES ARE SELECTABLE FOR GRAPHING.


PROCESS FIELD TERMINATOR
```

```
SUBROUTINE GETTIMET(LOSS,UNITID,SUBTI,SUMDATA,NUMUNIT,B,R,RAT4

PURPOSE:  TO READ TIME T NODE MESSAGE DATA FILE AND TIME T
   COMBAT LOSSES DATA FILE.

PARAMETERS:
  LOSS     => ARRAY CONTAINING COMBAT LOSSES FOR EACH UNIT
  UNITID   => ARRAY CONTAINING UNIT NAMES AND NUMBERS
  SUBTI    => SUBTITLE TO PRINT WITH GRAPH
  SUMDATA  => ARRAY CONTAINING MESSAGE DATA FOR EACH UNIT
  NUMUNIT  => NUMBER OF UNITS TO CHOOSE FROM
  B        => ARRAY OF BLUE LOSSES FOR ONE UNIT FOR ONE TIME
              INCREMENT
  R        => ARRAY OF RED LOSSES FOR ONE UNIT FOR ONE TIME
              INCREMENT
  RAT4     => BLUE ON RED FORCE RATIO
  FILE1    => FILE TO READ NODE MESSAGE DATA FROM
  FILE2    => FILE TO READ COMBAT LOSSES DATA FROM
  TIMET    => ARRAY CONTAINING THE TIME VALUES CORRESPONDING
              TO EACH TIME INCREMENT FOR NODE MESSAGE GRAPHS
  TIME     => NUMBER OF TIME INCREMENTS TO GRAPH FOR NODE
              MESSAGE GRAPHS
  TIMET2   => ARRAY CONTAINING THE TIME VALUES CORRESPONDING
              TO EACH TIME INCREMENT TO GRAPH FOR COMBAT
              LOSSES GRAPHS
  TIME2    => NUMBER OF TIME INCREMENTS TO GRAPH FOR COMBAT
              LOSSES GRAPHS
  MAXSYS   => TOTAL NUMBER OF COMBAT SYSTEMS TO CHOOSE FROM
  SAVEOLD2 =>  BUFFER USED TO COMPUTE INCREMENTAL LOSSES FROM
               CUMULATIVE LOSSES
  SAVENEW2 =>  BUFFER USED TO COMPUTE INCREMENTAL LOSSES FROM
               CUMULATIVE LOSSES
FMS TERMINATOR CODES

CONSTANTS

ALLOW USER TO CHANGE DEFAULT FILE NAMES TO LOAD FROM


PROCESS FIELD TERMINATOR

OPEN FILES


INITIALIZE WORKING AREA

READ TIME T DATA FILE
   TIME T DATA FILE CONSISTS OF MULTIPLE TIME T DATA SETS


SKIP OVER THE PREAMBLE DOCUMENTATION
```

READ IN INITIAL TIME T DATA SET WHICH CONTAINS PREVIOUS TOTALS. THESE ARE NEEDED TO COMPUTE THE FIRST INCREMENTAL CHANGE SINCE THE VALUES ARE READ IN AS CUMULATIVE VALUES. SKIP OVER HEADER

READ IN THE TIME CORRESPONDING TO THIS TIME T DATA SET

READ IN TIME T DATA SET

VALUES READ IN ARE RUNNING TOTALS AT TIME T. WE WANT TO HAVE THE VALUES FOR EACH INDIVIDUAL TIME T INCREMENT. THEREFORE, SUBTRACT THE PREVIOUS TIME T TOTAL.

THE LAST RECORD IN A TIME T DATA SET IS THE UNIT WHOSE IDENTIFIER IS EQUAL TO ONE.

READ IN INITIAL LOSSES DATA SET.
INITIAL DATA SET IS ALL ENTRIES WITH TIME VALUES OF ZERO.

FIND LOCATION OF UNIT NAME WITHIN ARRAY UNITID

FIND LOCATION OF UNIT NAME WITHIN ARRAY UNITID

SUBROUTINE GETUNIT(UNIT, NUMUNIT)

PURPOSE: DETERMINE WHICH UNIT WILL BE GRAPHED. FIND THE POSITION OF THE UNIT WITHIN THE DATA STRUCTURE.

PARAMETERS:

UNIT => POINTER TO WHICH UNIT TO GRAPH
NUMUNIT => NUMBER OF UNITS TO CHOOSE FROM

FMS TERMINATOR CODES

PROCESS FIELD TERMINATOR

SUBROUTINE GETVECTOR(VECTOR, NUMELEM, NUMUNIT)

PURPOSE: DETERMINE WHICH UNITS WILL BE GRAPHED. FIND THE POSITION OF EACH UNIT WITHIN THE DATA STRUCTURE AND STORE THE LOCATIONS IN A VECTOR. SUBROUTINE GETVECTOR RETURNS THE LOCATION VECTOR AND THE NUMBER OF ELEMENTS IN THE VECTOR.

PARAMETERS:

```
  VECTOR  =>  ARRAY OF POINTERS TO WHICH UNITS TO GRAPH
  NUMELEM =>  NUMBER OF UNITS CHOOSEN TO BE GRAPHED
  NUMUNIT =>  NUMBER OF UNITS TO CHOOSE FROM
```

FMS TERMINATOR CODES

PROCESS FIELD TERMINATOR

SUBROUTINE GRAPH1(SUBTI,UNITID,UNIT,SUMDATA,TIMET,TIME)

PURPOSE: CREATE DATA FILE FOR LINE GRAPH OF COMMUNICATIONS PATH LIMITS. GRAPH HAS LINES FOR NUMBER OF MESSAGES IN, OUT, HELD AND DELETED WITH RESPECT TO TIME.

PARAMETERS:

```
  SUBTI   => SUBTITLE TO PRINT WITH GRAPH
  UNITID  => ARRAY CONTAINING UNIT NAMES AND NUMBERS
  UNIT    => POINTER TO UNIT TO BE GRAPHED
  SUMDATA => ARRAY CONTAINING MESSAGE DATA FOR EACH UNIT
  TIMET   => ARRAY CONTAINING THE TIME VALUES CORRESPONDING
             TO EACH TIME INCREMENT
  TIME    => NUMBER OF TIME INCREMENTS TO GRAPH
```

CREATES INSTRUCTION PORTION OF DATA FILE.

CREATES DATA PORTION OF DATA FILE. EACH DATA LINE CONSISTS OF THE TIME AND THE NUMBER OF MESSAGES IN, OUT, HELD AND DELETED.

```
SUBROUTINE GRAPH2(SUBTI,UNITID,UNIT,SUMDATA,TIMET,TIME)

PURPOSE:   CREATE DATA FILE FOR LINE GRAPH OF INPUT MESSAGE
   LIMITS.  GRAPH HAS LINES FOR NUMBER OF MESSAGES IN,
   HELD AND DELETED WITH RESPECT TO TIME.

PARAMETERS:
  SUBTI   => SUBTITLE TO PRINT WITH GRAPH
  UNITID  => ARRAY CONTAINING UNIT NAMES AND NUMBERS
  UNIT    => POINTER TO UNIT TO BE GRAPHED
  SUMDATA => ARRAY CONTAINING MESSAGE DATA FOR EACH UNIT
  TIMET   => ARRAY CONTAINING THE TIME VALUES CORRESPONDING
             TO EACH TIME INCREMENT
  TIME    => NUMBER OF TIME INCREMENTS TO GRAPH


CREATES INSTRUCTION PORTION OF DATA FILE.


   CREATES DATA PORTION OF DATA FILE.  EACH DATA LINE
CONSISTS OF THE TIME AND THE NUMBER OF MESSAGES IN, HELD
AND DELETED.
```

SUBROUTINE GRAPH3(SUBTI, UNITID, UNIT, SUMDATA, TIMET, TIME)

PURPOSE: CREATE DATA FILE FOR LINE GRAPH OF OUTPUT MESSAGE LIMITS. GRAPH HAS LINES FOR NUMBER OF MESSAGES OUT, HELD AND DELETED WITH RESPECT TO TIME.

PARAMETERS:

```
  SUBTI   => SUBTITLE TO PRINT WITH GRAPH
  UNITID  => ARRAY CONTAINING UNIT NAMES AND NUMBERS
  UNIT    => POINTER TO UNIT TO BE GRAPHED
  SUMDATA => ARRAY CONTAINING MESSAGE DATA FOR EACH UNIT
  TIMET   => ARRAY CONTAINING THE TIME VALUES CORRESPONDING
             TO EACH TIME INCREMENT
  TIME    => NUMBER OF TIME INCREMENTS TO GRAPH
```

CREATES INSTRUCTION PORTION OF DATA FILE.

CREATES DATA PORTION OF DATA FILE. EACH DATA LINE CONSISTS OF THE TIME AND THE NUMBER OF MESSAGES OUT, HELD AND DELETED.

```
SUBROUTINE GRAPH4(SUBTI, UNITID, UNIT, SUMDATA, TIMET, TIME)

PURPOSE:  CREATE DATA FILE FOR LINE GRAPH OF COMBAT SORTIES.
   GRAPH HAS LINES FOR NUMBER OF CLOSE AIR SUPPORT AND
   HELICOPTER SORTIES WITH RESPECT TO TIME.

PARAMETERS:
  SUBTI   => SUBTITLE TO PRINT WITH GRAPH
  UNITID  => ARRAY CONTAINING UNIT NAMES AND NUMBERS
  UNIT    => POINTER TO UNIT TO BE GRAPHED
  SUMDATA => ARRAY CONTAINING MESSAGE DATA FOR EACH UNIT
  TIMET   => ARRAY CONTAINING THE TIME VALUES CORRESPONDING
             TO EACH TIME INCREMENT
  TIME    => NUMBER OF TIME INCREMENTS TO GRAPH


CREATES INSTRUCTION PORTION OF DATA FILE.


   CREATES DATA PORTION OF DATA FILE.  EACH DATA LINE
CONSISTS OF THE TIME AND THE NUMBER OF CLOSE AIR SUPPORT
AND HELICOPTER SORTIES.
```

```
SUBROUTINE GRAPH5(SUBTI.UNITID.UNIT.LOSS.COMSYS.NUMSYS.SIDE.

PURPOSE:
   CREATE DATA FILE FOR LINE GRAPH OF LOSSES.  GRAPH HAS LINES
FOR THE NUMBER OF RED OR BLUE LOSSES FOR THE SPECIFIED COMBAT
SYSTEMS AT THE SPECIFIED NODE.

  SUBTI    => SUBTITLE TO PRINT WITH GRAPH
  UNITID   => ARRAY CONTAINING UNIT NAMES AND NUMBERS
  UNIT     => POINTER TO UNIT TO BE GRAPHED
  LOSS     => ARRAY CONTAINING COMBAT LOSSES FOR EACH UNIT
  COMSYS   => ARRAY CONTAINING COMBAT SYSTEMS TO BE GRAPHED
  NUMSYS   => NUMBER OF COMBAT SYSTEMS TO BE GRAPHED
  SIDE     => WHICH SIDE(S) TO GRAPH
              1  ->  GRAPH BLUE
              2  ->  GRAPH RED
              3  ->  GRAPH BLUE & RED
  TIMET2   => ARRAY CONTAINING THE TIME VALUES CORRESPONDING
              TO EACH TIME INCREMENT
  TIME2    => NUMBER OF TIME INCREMENTS TO GRAPH
  MAXSYS   => TOTAL NUMBER OF COMBAT SYSTEMS TO CHOOSE FROM


CREATES INSTRUCTION PORTION OF DATA FILE


   CREATES LEGEND FOR GRAPHS WITH ONLY ONE SIDE


   CREATES DATA PORTION FOR GRAPHS WITH ONLY ONE SIDE


   CREATES LEGEND FOR GRAPHS WITH BOTH BLUE & RED SIDES


   CREATES DATA PORTION FOR GRAPHS WITH BOTH BLUE & RED SIDES
```

SUBROUTINE GRAPH6(SUBTI, UNITID, VECTOR, SUMDATA, NUMELEM, ITI

PURPOSE: CREATE DATA FILE FOR BAR GRAPH OF COMMUNICATIONS PATH LIMITS. GRAPH HAS BARS FOR THE NUMBER OF MESSAGES IN, OUT, HELD, AND DELETED FOR EACH UNIT IN THE VECTOR.

PARAMETERS:
SUBTI => SUBTITLE TO PRINT WITH GRAPH
UNITID => ARRAY CONTAINING UNIT NAMES AND NUMBERS
VECTOR => ARRAY OF POINTERS TO UNITS TO BE GRAPHED
SUMDATA => ARRAY CONTAINING MESSAGE DATA FOR EACH UNIT
NUMELEM => NUMBER OF UNITS TO BE GRAPHED
ITIME => TIME CORRESPONDING TO SUMMARY DATA

CONVERT TIME WHICH IS STORED AS AN INTEGER IN 1/2 HOUR INCREMENTS TO A CHARACTER STRING IN 1 HOUR INCREMENTS

CREATES INSTRUCTION PORTION OF DATA FILE.

CREATES DATA PORTION OF DATA FILE. EACH DATA LINE CONSISTS OF THE UNIT NAME IN QUOTES AND THE NUMBER OF MESSAGES IN, OUT, HELD AND DELETED.

SUBROUTINE GRAPH7(SUBTI, UNITID, VECTOR, SUMDATA, NUMELEM, ITI

PURPOSE: CREATE DATA FILE FOR BAR GRAPH OF INPUT MESSAGE LIMITS. GRAPH HAS BARS FOR THE NUMBER OF MESSAGES IN, HELD, AND DELETED FOR EACH UNIT IN THE VECTOR.

PARAMETERS:

```
  SUBTI   => SUBTITLE TO PRINT WITH GRAPH
  UNITID  => ARRAY CONTAINING UNIT NAMES AND NUMBERS
  VECTOR  => ARRAY OF POINTERS TO UNITS TO BE GRAPHED
  SUMDATA => ARRAY CONTAINING MESSAGE DATA FOR EACH UNIT
  NUMELEM => NUMBER OF UNITS TO BE GRAPHED
  ITIME   => TIME CORRESPONDING TO SUMMARY DATA
```

CONVERT TIME WHICH IS STORED AS AN INTEGER IN 1 2 HOUR INCREMENTS TO A CHARACTER STRING IN 1 HOUR INCREMENTS

CREATES INSTRUCTION PORTION OF DATA FILE.

CREATES DATA PORTION OF DATA FILE. EACH DATA LINE CONSISTS OF THE UNIT NAME IN QUOTES AND THE NUMBER OF MESSAGES IN, HELD AND DELETED.

SUBROUTINE GRAPH8(SUBTI, UNITID, VECTOR, SUMDATA, NUMELEM, ITI

PURPOSE: CREATE DATA FILE FOR BAR GRAPH OF OUTPUT MESSAGE LIMITS. GRAPH HAS BARS FOR THE NUMBER OF MESSAGES OUT, HELD, AND DELETED FOR EACH UNIT IN THE VECTOR.

PARAMETERS:

- SUBTI => SUBTITLE TO PRINT WITH GRAPH
- UNITID => ARRAY CONTAINING UNIT NAMES AND NUMBERS
- VECTOR => ARRAY OF POINTERS TO UNITS TO BE GRAPHED
- SUMDATA => ARRAY CONTAINING MESSAGE DATA FOR EACH UNIT
- NUMELEM => NUMBER OF UNITS TO BE GRAPHED
- ITIME => TIME CORRESPONDING TO THE SUMMARY DATA

CONVERT TIME WHICH IS STORED AS AN INTEGER IN 1/2 HOUR INCREMENTS TO A CHARACTER STRING IN 1 HOUR INCREMENTS

CREATES INSTRUCTION PORTION OF DATA FILE.

CREATES DATA PORTION OF DATA FILE. EACH DATA LINE CONSISTS OF THE UNIT NAME IN QUOTES AND THE NUMBER OF MESSAGES OUT, HELD AND DELETED.

SUBROUTINE GRAPH9(SUBTI, UNITID, VECTOR, SUMDATA, NUMELEM, ITI

PURPOSE: CREATE DATA FILE FOR BAR GRAPH OF COMBAT SUPPORT. GRAPH HAS BARS FOR THE NUMBER OF CLOSE AIR SUPPORT AND HELICOPTER SORTIES FOR EACH UNIT IN THE VECTOR.

PARAMETERS:

- SUBTI - SUBTITLE TO PRINT WITH GRAPH
- UNITID - ARRAY CONTAINING UNIT NAMES AND NUMBERS
- VECTOR - ARRAY OF POINTERS TO UNITS TO BE GRAPHED
- SUMDATA - ARRAY CONTAINING MESSAGE DATA FOR EACH UNIT
- NUMELEM - NUMBER OF UNITS TO BE GRAPHED
- ITIME - TIME CORRESPONDING TO THE SUMMARY DATA

CONVERT TIME WHICH IS STORED AS AN INTEGER IN 1/2 HOUR INCREMENTS TO A CHARACTER STRING IN 1 HOUR INCREMENTS

CREATES INSTRUCTION PORTION OF DATA FILE.

CREATES DATA PORTION OF DATA FILE. EACH DATA LINE CONSISTS OF THE UNIT NAME IN QUOTES AND THE NUMBER OF CLOSE AIR SUPPORT AND HELICOPTER SORTIES.

SUBROUTINE GRAPH10(SUBTI,UNITID,VECTOR,LOSS,COMSYS,NUMSYS,MAXS

PURPOSE: CREATE DATA FILE FOR BAR GRAPH OF LOSSES. GRAPH HAS BARS FOR THE NUMBER OF BLUE AND RED LOSSES FOR EACH COMBAT SYSTEM FOR THE SPECIFIED UNIT.

PARAMETERS:

SUBTI	=>	SUBTITLE TO PRINT WITH GRAPH
UNITID	=>	ARRAY CONTAINING UNIT NAMES AND NUMBERS
VECTOR	=>	ARRAY OF POINTERS TO UNITS TO BE GRAPHED
LOSS	=>	ARRAY CONTAINING COMBAT LOSSES FOR EACH UNIT
COMSYS	=>	ARRAY CONTAINING COMBAT SYSTEMS TO BE GRAPHED
NUMSYS	=>	NUMBER OF COMBAT SYSTEMS TO BE GRAPHED
MAXSYS	=>	TOTAL NUMBER OF COMBAT SYSTEMS TO CHOOSE FROM
ITIME	=>	TIME CORRESPONDING TO THE SUMMARY DATA

CONVERT TIME WHICH IS STORED AS AN INTEGER IN 1/2 HOUR INCREMENTS TO A CHARACTER STRING IN 1 HOUR INCREMENTS

CREATES INSTRUCTION PORTION OF DATA FILE.

CREATES DATA PORTION OF DATA FILE. EACH DATA LINE CONSISTS OF THE NAME OF THE COMBAT SYSTEM AND THE BLUE AND RED LOSSES AT THE SPECIFIED UNIT.

SUBROUTINE OPTIONS(SCREEN, PRINT)

PURPOSE: ALLOWS THE USER TO SELECT WHICH TYPE OF GRAPH IS TO BE CREATED AND WHETHER THE GRAPH IS TO BE PRINTED OR JUST VIEWED ON THE SCREEN.

PARAMETERS:

```
  SCREEN    => USER'S SELECTION OF WHICH GRAPH TO CREATE
  PRINT     => PRINT FLAG
                 0 -> DO NOT PRINT
                 1 -> PRINT
```

FMS TERMINATOR CODES

PROCESS FIELD TERMINATOR

```
SUBROUTINE SCRBK(FLDNAM, CURPOS, LINE, FVALUE)

PURPOSE:    SCROLL BACKWARD THE SCROLLED AREA FOR UNITS

PARAMETERS:
   FLDNAM  ==> NAME OF FIELD IN SCROLLED AREA TO SCROLL
   CURPOS  ==> ARRAY INDICE OF CURRENT ENTRY
   LINE    ==> LINE NUMBER OF THE CURRENT LINE OF THE
               SCROLLED AREA.
   FVALUE  ==> LINE TO OUTPUT TO SCROLLED AREA IF CURRENT LINE
               IS THE TOP LINE OF THE SCROLLED AREA

FMS TERMINATOR CODES
```

SUBROUTINE SCRBK2(FLDNAM, CURPOS, LINE, FVALUE, PREV)

PURPOSE: SCROLL BACKWARD THE SCROLLED AREA FOR UNITS WHERE THE ONLY UNITS USED ARE THOSE WHO OBTAINED COMBAT LOSSES.

PARAMETERS:

```
   FLDNAM  ==> NAME OF FIELD IN SCROLLED AREA TO SCROLL
   CURPOS  ==> ARRAY INDICE OF CURRENT ENTRY
   LINE    ==> LINE NUMBER OF THE CURRENT LINE OF THE
               SCROLLED AREA.
   FVALUE  ==> LINE TO OUTPUT TO SCROLLED AREA IF CURRENT LINE
               IS THE TOP LINE OF THE SCROLLED AREA
   PREV    ==> ARRAY INDICE OF PREVIOUS ENTRY
```

FMS TERMINATOR CODES

SUBROUTINE SCRFWD(FLDNAM, MAXLINE, CURPOS, LINE, MAXARRAY, FVA

PURPOSE: SCROLL FORWARD THE SCROLLED AREA FOR UNITS

PARAMETERS:

```
   FLDNAM   ==> NAME OF FIELD IN SCROLLED AREA TO SCROLL
   MAXLINE  ==> NUMBER OF LINES IN THE SCROLLED AREA
   CURPOS   ==> ARRAY INDICE OF CURRENT UNIT
   LINE     ==> LINE NUMBER OF THE CURRENT LINE OF THE
                SCROLLED AREA.
   MAXARRAY => MAXIMUM VALUE THAT VARIABLE CURPOS CAN OBTAIN
   FVALUE   ==> LINE TO OUTPUT TO SCROLLED AREA IF CURRENT LINE
                IS THE TOP LINE OF THE SCROLLED AREA
```

FMS TERMINATOR CODES

```
SUBROUTINE SCRFWD2(FLDNAM, MAXLINE, CURPOS, LINE, MAXARRAY,

PURPOSE:   SCROLL FORWARD THE SCROLLED AREA FOR UNITS
   WHERE THE ONLY UNITS USED ARE THOSE WHO OBTAINED
   COMBAT LOSSES.

PARAMETERS:
   FLDNAM   ==> NAME OF FIELD IN SCROLLED AREA TO SCROLL
   MAXLINE  ==> NUMBER OF LINES IN SCROLLED AREA
   CURPOS   ==> ARRAY INDICE OF CURRENT ENTRY
   LINE     ==> LINE NUMBER OF THE CURRENT LINE OF THE
                SCROLLED AREA.
   MAXARRAY => MAXIMUM VALUE WHICH VARIABLE CURPOS CAN OBTAIN
   FVALUE   ==> LINE TO OUTPUT TO SCROLLED AREA IF CURRENT LINE
                IS THE TOP LINE OF THE SCROLLED AREA
   NEXT     ==> ARRAY INDICE OF NEXT ENTRY


FMS TERMINATOR CODES
```

```
SUBROUTINE SCRLINE(NEXT,MAX,FVALUE)

PURPOSE:  CREATE LINE TO OUTPUT TO SCROLLED AREA FOR UNITS

PARAMETERS:
  NEXT    => ARRAY INDICE OF UNIT TO DISPLAY
  MAX     => MAXIMUM LEGAL VALUE WITHIN ARRAY CONTAINING UNITS
  FVALUE  => CHARACTER STRING TO OUTPUT TO SCROLLED AREA


   NULL ENTRY, THEREFORE, INITIALIZE TO BLANKS
```

```
SUBROUTINE SCRLINE2 (NAME,FLAG,NEXT,MAXSYS,FVALUE)

PURPOSE:  CREATE LINE TO OUTPUT TO SCROLLED AREA FOR COMBAT SY

PARAMETERS:
  NEXT     => ARRAY INDICE OF COMBAT SYSTEM TO DISPLAY
  MAX      => MAXIMUM LEGAL VALUE WITHIN ARRAY CONTAINING COMBA
              SYSTEM
  FVALUE   => CHARACTER STRING TO OUTPUT TO SCROLLED AREA


   NULL ENTRY, THEREFORE, INITIALIZE TO BLANKS
```

SUBROUTINE SCRLINE3(NEXT,MAX,FVALUE)

PURPOSE: CREATE LINE TO OUTPUT TO SCROLLED AREA FOR UNITS
WHO SUSTAINED COMBAT LOSSES.

PARAMETERS:
NEXT   => ARRAY INDICE OF UNIT TO DISPLAY
MAX   => MAXIMUM LEGAL VALUE WITHIN ARRAY CONTAINING UNITS
FVALUE => CHARACTER STRING TO OUTPUT TO SCROLLED AREA

SKIP OVER ANY UNIT'S WHOSE LOSSES FLAG INDICATES NO LOSSES

NULL ENTRY, THEREFORE, INITIALIZE TO BLANKS

****** UAR ROUTINES **********

INTEGER FUNCTION VALID1

FIELD COMPLETION UAR CHECKS IF VALUE IS BETWEEN 1
AND THE MAXIMUM FOR THE APPROPRIATE FIELD.

GET FIELD NAME OF CURRENT FIELD
GET VALUE AT CURRENT FIELD
CONVERT VALUE TO INTEGER
GET MAXIMUM LEGAL VALUE FOR CURRENT FIELD
CONVERT MAXIMUM LEGAL VALUE TO INTEGER
IF CURRENT VALUE IS BETWEEN 1 AND MAXIMUM LEGAL VALUE
 THEN RETURN SUCCESS CODE ELSE SEND ERROR MESSAGE AND
 RETURN FAILURE CODE.

Distribution

IDA Paper P-1978

C3EVAL MODEL DEVELOPMENT--1986, Volume II: Programmers' Manual

75 Copies

DEPARTMENT OF DEFENSE	Copies
Office Joint Chiefs of Staff Washington, DC 20301-5000	
Attn: Dr. Robert J. Fallon, J-6A	12
Dr. John Dockery, J-6	1
Mr. Vince P. Roske, J-8	1
COL James J. Sidletsky, J-8	2
CPT W. L. Butler, USN, J-3	1
LTC J. P. Morrison, USAF, J-4	1
CDR T. R. Sheffield, USN, J-5	1
Directorate of Information and Resource Management	20
Office of the Under Secretary of Defense for Research & Engineering Room 3D139, Pentagon Washington, DC 20310	
Attn: Mr. James D. Bain, C3I	1
Director Defense Intelligence Agency Washington, DC 20301-6111	
Attn: Mr. A. J. Straub, Pomponio Plaza 1023	1
Office of the Secretary of Defense OUSDRE (DoD-IDA Management Office) 1801 North Beauregard Street Alexandria, Virginia 22311	
Attn: COL John P. Wilhelm	1
Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2

DEPARTMENT OF THE ARMY

Deputy Chief of Staff for Operations and Plans
Department of the Army
Room 3C542, Pentagon
Washington, DC 20310-0430

Attn:	MAJ W. E. Ward, USA	1
	Mr. Hunter Woodall, DCS/RDA	1

Director
Department of the Army
Communications Electronic Command
Fort Monmouth, NJ 07703-5207

Attn:	Mr. Michael Horvath AMSEL-RD-ASCO-SE	1

DEPARTMENT OF THE NAVY

Chief of Naval Operations
Department of the Navy
Room 4E549, Pentagon
Washington, DC 20350

Attn:	CDR D. L. McKinney, USN NOP	1

Commander
Naval Postgraduate School
Monterey, CA 93940

Attn:	Prof Michael G. Sovereign, Chairman, Command, Control and Communications	2

DEPARTMENT OF THE AIR FORCE

Deputy Chief of Staff
Operations, Plans and Readiness
Department of the Air Force
Washington, DC 20330-6600

Attn:	COL R. C. McFarlane, AFXOXR	1

Headquarters
US Marine Corps
Columbia Pike and South Arlington Ridge Road
Arlington, VA 22204

Attn:	LTC T. L. Wilkerson, Office of Deputy Chief of Staff, Plans, Policy and Operations (MD-P)	1

INDUSTRIAL ORGANIZATIONS

Applications Research Corporation 330 South Ludlow Street Dayton, OH 45402		
Attn: Mr. Rodney B. Beach		1
Institute for Defense Analyses 1801 North Beauregard Street Alexandria, Virginia 22311		22
Attn: Gen W. Y. Smith	1	
Mr. Seymour Deitchman	1	
Mr. A. R. Barbeau	1	
Dr. William. J. Schultis	1	
Mr. Robert F. Robinson	1	
Mr. Edward Kerlin	1	
Mr. J. L. Freeh	1	
Dr. Donald Ockerman	1	
Dr. E. Simaitis	1	
Mr. Joseph W. Stahl	1	
Dr. V. A. Utgoff	1	
Control and Distribution	11	

END
12-87
DTIC